VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV INFORMATIKY
INSTITUTE OF INFORMATICS

NÁVRH A TVORBA PROXY PRO PENETRAČNÍ TESTOVÁNÍ
DESIGN AND CREATION OF PROXY FOR PENETRATION TESTING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Michal Válka
AUTHOR

VEDOUCÍ PRÁCE — Ing. Petr Dydowicz, Ph.D.
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA PODNIKATELSKÁ

Zadání bakalářské práce

Ústav:	Ústav informatiky
Student:	**Michal Válka**
Studijní program:	Systémové inženýrství a informatika
Studijní obor:	Manažerská informatika
Vedoucí práce:	**Ing. Petr Dydowicz, Ph.D.**
Akademický rok:	2019/20

Ředitel ústavu Vám v souladu se zákonem č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a se Studijním a zkušebním řádem VUT v Brně zadává bakalářskou práci s názvem:

Návrh a tvorba proxy pro penetrační testování

Charakteristika problematiky úkolu:

Úvod
Vymezení problému a cíle práce
Teoretická východiska práce
Analýza problému a současné situace
Vlastní návrh řešení, přínos práce
Závěr
Seznam použité literatury

Cíle, kterých má být dosaženo:

Bakalářská práce se zabývá návrhem a tvorbou proxy pro penetrační testování aplikací. Cílem práce je zanalyzovat současnou situaci a vymezit problémy, se kterými se bezpečnostní specialisté potýkají při používání nástrojů sloužících k penetračnímu testování aplikací. Na základě provedených analýz budou definovány požadavky na nové softwarové řešení, jež bude podle sestavených požadavků navrženo a vytvořeno tak, aby usnadnilo provádění penetračních testů a rozšířilo možnosti penetračních testerů.

Základní literární prameny:

BASL, J. a R. BLAŽÍČEK. Podnikové informační systémy. Podnik v informační společnosti. Praha: Grada, 2008. 283 s. ISBN 978-80-247-2279-5.

MOLNÁR, Z. Automatizované informační systémy. Praha: Strojní fakulta ČVUT, 2000. 126 s. ISBN 80-01-02269-2.

MOLNÁR, Z. Efektivnost informačních systémů. Praha: Grada Publishing, 2000. 142 s. ISBN 80-716--410-X.

PECINOVSKÝ, R. Myslíme objektově v jazyku Java: kompletní učebnice pro začátečníky. Praha: Grada, 2009. 570 s. ISBN 978-80-247-2653-3.

SODOMKA, P. a H. KLČOVÁ. Informační systémy v podnikové praxi. Brno: Computer Press, 2010. 501 s. ISBN 978-80-251-2878-7.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2019/20

V Brně dne 29.2.2020

L. S.

doc. RNDr. Bedřich Půža, CSc. ředitel	doc. Ing. et Ing. Stanislav Škapa, Ph.D. děkan

Abstrakt

Bakalářská práce se zaměřuje na návrh a vývoj proxy sloužící pro účely penetračního testování. Práce se dělí na tři hlavní části a začíná teoretickou částí rozebírající základní technologie a principy, na kterých aplikace staví. Druhá kapitola se zabývá analýzou současného stavu hodnotící aktuálně dostupné alternativy pro penetrační testy síťové komunikace. Z provedených analýz jsou vyvozeny požadavky na finální produkt, jehož návrh a vývoj je popsán v kapitole vlastního návrhu. V závěru práce je shrnuto vyvinuté řešení a jeho přínosy pro penetrační testování.

Klíčová slova

softwarová aplikace, penetrační testování, proxy, aplikační bezpečnost, hacking, síťová komunikace, java

Abstract

This bachelor's thesis is aimed at design and development of proxy for penetration testing. The thesis is divided into three main parts and begins with a theoretical part, which is focused on fundamental technologies and principles on which the application is based. The second part is focused on comparison of currently available solutions. The third part contains the creation of the proxy itself. The last chapter contains a summary of this thesis and the benefits of the developed product for penetration testing.

Key words

software application, penetration testing, proxy, software security, hacking, network communication, java

Bibliografická citace

VÁLKA, Michal. *Návrh a tvorba proxy pro penetrační testování* [online]. Brno, 2020 [cit. 2020-05-05]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/127503. Bakalářská práce. Vysoké učení technické v Brně, Fakulta podnikatelská, Ústav informatiky. Vedoucí práce Petr Dydowicz.

Čestné prohlášení

Prohlašuji, že předložená bakalářská práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 5. května 2020

……

podpis autora

Poděkování

Rád bych poděkoval panu Ing. Petru Dydowiczovi, Ph.D. za vedení této závěrečné práce, své rodině i přátelům za jejich podporu, a také svým kolegům za pomoc při testování aplikace v reálném prostředí.

OBSAH

ÚVOD

V současné době se bezpečnost v oblasti informačních technologií řeší na denním pořádku a na penetrační testy prověřující kvalitu zabezpečení konkrétních aplikací podniky vynakládají nemalé finanční prostředky. Nejčastěji jsou prováděny penetrační testy webových aplikací, avšak postupem času se zvyšuje zájem i o testy jiných síťových aplikací. S tímto zájmem a stále složitějšími technologiemi roste poptávka po kvalitních nástrojích, které usnadní vyhledávání zranitelností v citlivých službách.

Cílem této práce je vyvinutí proxy, jež rozšíří možnosti penetračních testerů a umožní kvalitnější a rychlejší analýzu síťové komunikace testovaných aplikací. Koncepce proxy bude navíc navržena tak, aby bylo možné proxy používat dohromady s dalšími nástroji určenými pro práci s datovými přenosy.

V kapitole teoretických východisek bude rozebráno teoretické pozadí problematiky, které je základem pro celou aplikaci programovanou v rámci této práce. Zejména jde o vysvětlení použitých technologií a principů, na kterých vyvíjená testovací proxy staví.

V kapitole třetí bude provedena analýza současného stavu a bude kladen důraz na aktuální možnosti, jež mají etičtí i neetičtí hackeři k dispozici při útocích na aplikace v oblasti modifikace komunikace. Součástí této kapitoly bude také rozbor existujících nástrojů a zhodnocení jejich výhod a nevýhod pro praktické použití.

V kapitole vlastních návrhů řešení bude podrobně rozebrán návrh architektury i uživatelského prostředí aplikace, vývoj aplikačního jádra a vývoj jednotlivých interních modulů umožňujících práci s daty procházejícími přes proxy. Konec kapitoly bude věnován ekonomickému zhodnocení a přínosům vytvořeného řešení.

VYMEZENÍ PROBLÉMU A CÍLE PRÁCE

V následujících dvou podkapitolách rozeberu cíle práce a metody, jež budou využity k dosažení vytyčeného cíle.

Cíl práce

Cílem bakalářské práce je naprogramovat multiplatformní proxy pro účely penetračního testování, která může být využita při hledání bezpečnostních chyb ve službách komunikujících prostřednictvím síťové komunikace.

Výsledná proxy by měla fungovat jako nástroj usnadňující práci se síťovým provozem aplikací a měla by být využitelná při praktickém testování bezpečnostních zranitelností.

Metody a postupy zpracování

Při zpracování bakalářské práce se nejdříve zaměřím na aktuální nástroje a metodiky používané při testování síťových aplikací a následně provedu návrh vlastního řešení, které bude cílit na nedostatky existujících řešení.

Samotné vytváření aplikace bude započato návrhem uživatelského prostředí a architektury jádra nástroje. Další vývoj se pak bude zaměřovat na konkrétní moduly, jež aplikaci rozšiřují o konkrétní funkcionalitu usnadňující proces penetračního testování.

1 TEORETICKÁ VÝCHODISKA PRÁCE

V této kapitole se věnuji teoretickým východiskům, na kterých celá práce staví a bez nichž by nemohla vzniknout. Rozeberu zde nejdůležitější technologie a principy, a to především programovací jazyk Java, pojmy penetrační test a proxy, referenční model ISO/OSI, architekturu TCP/IP a komunikační protokoly HTTP a WebSockets.

1. 1 Objektově orientované programování

Objektově orientované programování je přístup k psaní kódu zakládající si na používání tzv. objektů, které reprezentují konkrétní prvky, s nimiž vývojář pracuje a jimž dává určité vlastnosti a definuje jejich chování. Celý princip objektově orientovaného programování vychází z lidského chápání objektů. Vzájemným provázáním objektů je pak tvořen výsledný funkční celek (1).

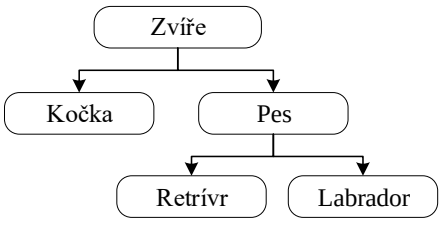
1. 1. 1 Tři principy OOP

Objektově orientované programování staví na třech základních principech:

- zapouzdření,
- dědičnosti,
- polymorfismu (1).

Zapouzdření je formou zabezpečení, které pevně definuje, jak lze přistupovat k jednotlivým metodám či vlastnostem objektů a tříd, což umožňuje vytvářet například určitou míru abstrakce mezi veřejným rozhraním a implementací a zároveň předchází vzniku mnohých chyb (1).

Dědičnost je základní princip zavádějící do objektově orientovaného programování hierarchii, v níž z nadřazeného prvku mohou vycházet prvky podřazené a rozšiřovat jeho vlastnosti o své vlastní (1).



Obrázek č. 1 Hierarchie v OOP
(Zdroj: Vlastní zpracování dle: 1)

Polymorfismus je vlastnost, která umožňuje nadefinovat nadřazené rozhraní obsahující obecné akce pro jednotlivé třídy, přičemž konkrétní implementace se v jednotlivých třídách mohou lišit. Vývojář při použití podřazených tříd pak nemusí znát konkrétní implementace jednotlivých akcí a stačí mu znalost nadřazeného rozhraní, čímž je dosaženo značeného usnadnění vývoje (1).

Ve výsledném celku, v podobě objektově orientovaného kódu, se zapouzdření, dědičnost a polymorfismus navzájem doplňují a dávají programátorovi silný nástroj pro budování rozsáhlých a dlouhodobě udržitelných aplikací (1).

1. 2 Java

Java je jeden z nejpoužívanějších programovacích jazyků současné doby. Vznikal jako reakce na poptávku po vývoji pro více operačních systémů současně, již v roce 1991. K vysoké popularitě Javy pak značně pomohl nárust zájmu o webové technologie, k jejichž tvorbě se Java používá dodnes, a to zejména pro svoji přenositelnost a komplexnost (1).

Celý princip nezávislosti Java aplikací na cílovém operačním systému je založen na rozdělení procesu pro zpracování zdrojového kódu na dvě samostatné části, a to na kompilaci do tzv. bajtkódu a jeho následnou interpretaci prostřednictvím JVM (Java Virtual Machine) (1).

Zdrojový kód aplikace je reprezentován množinou souborů s příponou .java, které obsahují člověkem čitelný kód, a výsledná zkompilovaná aplikace se skládá ze souborů s příponou .class, jež obsahují bajtkód interpretovatelný virtuálním strojem Javy (1).

Obrázek č. 2 Kompilace a interpretace Java aplikace
(Zdroj: Vlastní zpracování dle: 1)

Kvůli slabšímu výkonu interpretovaných aplikací Java v současné době využívá Just-In-Time kompilátoru, který dokáže dle potřeby jednotlivé části bajtkódu zkompilovat na spustitelný kód (1).

1. 2. 1 Výhody jazyka Java

Mezi hlavní výhody jazyka Java patří:

- jednoduchost – Java je snadno naučitelná,
- objektová orientovanost – Java byla navržena v souladu s OOP,
- přenositelnost – nezávislost na operačních systémech,
- robustnost – automatizovaná správa zdrojů,
- multi-threading – podpora tvorby vícevláknových aplikací,
- výkonnost – vysoký výkon výsledných aplikací,
- distribuovatelnost – podpora TCP/IP i vzdáleného volání metod,
- dynamičnost – lze dynamicky aktualizovat části bajtkódu (1).

1. 2. 2 Knihovna jazyka Java

Java je velmi komplexní programovací jazyk a lze jej uplatnit při vývoji serverových i klientských aplikací, přičemž v obou případech jsou pro programování k dispozici rozsáhlé balíčky knihovny Java, které usnadňují práci programátorovi a poskytují mu programové řešení pro různé problematiky jako je vytváření vláken a práce s nimi, reflexivní práce s objekty nebo práce s kolekcemi dat (1).

Mezi balíčky nacházející se v knihovně Java patří například:

- java.lang – základní prvky jazyka Java,
- java.util – kolekce, zpracování času a data, formátování, regulární výrazy, logování, paralelní programování a další,
- java.io – I/O operace jako práce se soubory a výstupem,
- java.nio – I/O operace s využitím kanálů a bufferů,
- java.net – síťová komunikace (1).

1. 2. 3 JavaFX

JavaFX je moderní grafická knihovna pro tvorbu multiplatformních GUI v programovacím jazyce Java. Výhodou oproti starším knihovnám je možnost definování grafického rozložení pomocí souborů ve formátu XML a stylování pomocí kaskádových stylů (2).

Struktura knihovny JavaFX je rozdělena do následujících sedmi modulů, které lze dle potřeby zahrnout v aplikaci:

- javafx.base – základní API,
- javafx.controls – jednotlivé ovládací prvky pro vstupy a výstupy uživatele,
- javafx.fxml – podpora XML šablon,
- javafx.graphics – API pro kontejnery, okna a další prvky grafiky,
- javafx.media – multimediální podpora,
- javafx.swing – modul pro integraci s knihovnou Swing,
- javafx.web – webový prvek (2).

1. 2. 4 Gson

Gson je knihovna vytvořená společností Google Inc. v roce 2008. Účel knihovny je usnadnění práce s formátem JSON, a to například za pomoci přímé serializace a deserializace Java objektů do a z JSON řetězců. Užitečné jsou také utility jako je automatické formátování řetězců do přehledné formy a pomocné objekty pro práci s JSON soubory (3).

1. 2. 5 Gradle

Gradle je nástroj pro sestavování programů v širokém spektru programovacích jazyků včetně Javy, který rozšiřuje vývojáři možnosti pro automatizaci procesů pomocí vytváření skriptů, správy závislostí, definic úkolů a mnohých dalších podpůrných funkcí. Nástroj je navíc rozšiřitelný o vlastní i komunitní pluginy a lze jej snadno integrovat do všech populárních vývojových prostředí (4).

1. 3 XML a FXML

Značkovací jazyk XML (z anglického eXtensible Markup Language) slouží k vytváření dokumentů složených z entit, které tvoří hierarchii začínající kořenovou entitou. Při psaní XML dokumentů se používají takzvané tagy, které mají tři základní podoby:

- <pocatecni_tag>,
- </koncovy_tag>,
- <tag_prazdneho_elementu /> (5).

```xml
<?xml version="1.0"?>
<koren>
    <tag_1>Hodnota 1</tag_1>
    <tag_2 atribut1="1234">Hodnota 2</tag_2>
    <tag_3 atribut2="hodnota" />
</koren>
```

Obrázek č. 3 Příklad XML souboru
(Zdroj: Vlastní zpracování dle: 5)

Knihovna JavaFX využívá značkovací jazyk XML se svým vlastním označením FXML pro návrh grafického rozhraní aplikací, který si zakládá na definování struktury grafických scén pomocí speciálních tagů zastupujících jednotlivé prvky GUI. Jde o praktickou alternativu ke generování rozhraní uvnitř kódu, která umožňuje napojit navrženou XML strukturu na zdrojový kód pomocí takzvaných Controllers a anotací s označením @FXML (2).

Controllers (ovladače, kontroléry) jsou třídy pro práci s jednotlivými grafickými jednotkami, jako jsou záložky nebo celý obsah okna aplikace, a jejich použitím mohou vývojáři zpracovávat vzniklé události a pracovat s konkrétními komponentami definovanými ve FXML souborech. K tomuto účelu je nutné pojmenovat pomocí jednoznačného identifikátoru jednotlivé komponenty v FXML souborech a ve zdrojovém kódu aplikovat anotaci @FXML pro provázání daných proměnných a metod (2).

1. 4 HTML

Značkovací jazyk HTML (z anglického Hypertext Markup Language) vznikl v roce 1990 a v současnosti je používán ve své páté verzi označované HTML5 především pro vytváření strukturovaných dokumentů během vývoje klientských sekcí webových aplikací (6).

HTML soubory obsahují počáteční definici typu dokumentu <!DOCTYPE html>, po níž následuje kořenový element stránky <html>. Obsah kořenového tagu se pak dělí na dvě základní části, a to hlavičku stránky <head> obsahující popisné informace a tělo stránky <body>, do kterého se zapisuje rozložení a obsah stránky (6).

```
<!DOCTYPE html>
<html lang="cs">
    <head>
        <title>Titulek</title>
    </head>
    <body>
        Obsah
    </body>
</html>
```

Obrázek č. 4 Příklad HTML souboru
(Zdroj: Vlastní zpracování dle: 6)

Standard HTML povoluje vývojářům tvorbu vlastních elementů, a přitom definuje velké množství různých tagů, které je možné použít, například:

- <h1>, …, <h6> – jednotlivé úrovně nadpisů,
- <p> – odstavce,
- <img> – obrázek,
- <button> – tlačítko,
- <table> – tabulka (6).

1. 5 CSS

Kaskádové styly CSS (z anglického Cascading Style Sheets) jsou jazyk sloužící k popisu způsobu zobrazení dokumentů pomocí pravidel skládajících se ze selektoru, který ukazuje na konkrétní element či skupinu elementů, a deklarací, které popisují konkrétní vlastnosti (7).

Jazyk CSS je v JavaFX možné použít pro stylování grafických scén, k němuž jsou na rozdíl od standardního CSS používány speciální vlastnosti začínající pomlčkou. Jednotlivé grafické prvky GUI jsou vybírány přes jejich třídní selektory (2).

```
.tab, .pane {
    -fx-background-color: rgb(245, 16, 43);
}
```

Obrázek č. 5 Příklad CSS v JavaFX
(Zdroj: Vlastní zpracování dle: 2)

1. 6 Počítačová síť

Počítačová síť propojuje jednotlivá zařízení prostřednictvím síťové infrastruktury, a to za účelem vzájemné výměny dat. V současnosti jsou počítačové sítě používány pro komunikaci rozličných prvků v domácnostech i firemním sektoru. Na jejich fungování závisí elektrárny, ministerstva, školy, banky a mnoho dalších institucí (8).

1. 7 Referenční model ISO/OSI

OSI (Open Systems Interconnection) je sedmivrstvý referenční síťový model vytvořený společností ISO (Interntional Organization for Standardization). Pro jeho přílišnou těžkopádnost se však stal především pomůckou pro pochopení síťové komunikace a referenčním slovníkem pro kategorizaci síťových zařízení (8).

Model se skládá z následujících sedmi vrstev, mezi kterými probíhá vertikální komunikace (v rámci jednoho síťového prvku) a horizontální komunikace (mezi více síťovými prvky):

1. fyzická vrstva,
2. linková vrstva,
3. síťová vrstva,
4. transportní vrstva,
5. relační vrstva,
6. prezentační vrstva,
7. aplikační vrstva (8).

Horizontální komunikace mezi protilehlými vrstvami si zakládá na zapouzdřování přenášených dat pomocí přidávání záhlaví při průchodu z vyšší vrstvy na vrstvu nižší a následného odpouzdřování při průchodu z nižší vrstvy na vyšší vrstvu. Horizontální komunikace umožňuje výměnu informací mezi jednotlivými vrstvami na oddělených zařízeních, aniž by musely vrstvy odlišných úrovní rozumět záhlaví ostatních vrstev (8).

1. 7. 1 Fyzická vrstva

Fyzická vrstva má za cíl přenos jednotlivých bitů po přenosových médiích, jimiž mohou být například optické a mechanické kabeláže či rádiové vlny. Pro jednotlivá přenosová média musí být na fyzické vrstvě vyřešen způsob reprezentace bitů a princip jejich výměny mezi dvěma propojenými zařízeními (8).

Na fyzické vrstvě fungují základní síťové prvky jako jsou opakovače, které slouží k prodloužení dosahu fyzického přenosového média, a rozbočovače, jež přijaté bity z jednoho přenosového média předávají na dvě a více připojených přenosových médií (8).

1. 7. 2 Linková vrstva

Linková vrstva se stará o přenos rámců složených ze záhlaví, dat a zápatí mezi dvěma zařízeními propojenými fyzickou vrstvou. Mezi její úkoly patří především označení začátku a konce rámce, aby bylo možné jednotlivé bity z fyzické vrstvy složit do rámců, regulace přenosové rychlosti, označení vysílací a přijímací strany příslušnými adresami, detekce kolizí, podpora VLAN a kontrola bezchybného přenosu rámců (8).

1. 7. 3 Síťová vrstva

Síťová vrstva se stará o přenos a směrování takzvaných paketů k cílovým uzlům. Může tedy jít o přenos v rámci menší interní sítě, ale i přenos přes půl zeměkoule. Pro odlišení jednotlivých síťových uzlů se v praxi používají adresy IPv4 a IPv6 spadající pod rodinu protokolů TCP/IP (8).

1. 7. 4 Transportní vrstva

Transportní vrstva slouží pro přenos datagramů mezi procesy odesílatele a příjemce. Jejím úkolem je oddělení aplikačně orientovaných vrstev (aplikační, prezentační a relační) od především hardwarově orientovaných vrstev (síťové, linkové a fyzické) (8).

1. 7. 5 Relační vrstva

Relační vrstva se stará o spojení – jeho navázání, udržování i ukončování. Ze sedmi vrstev ISO/OSI modelu jde o nejméně vytíženou vrstvu (8).

1. 7. 6 Prezentační vrstva

Prezentační vrstva zařizuje konverze mezi různými kódováními přenášených dat, aby jim rozuměly aplikace na různých operačních systémech a zařízeních. Může tak jít například o práci s kódováním znaků, šifrováním, kompresí, či formátem přenášených dat (8).

1. 7. 7 Aplikační vrstva

Aplikační vrstva zahrnuje velké množství softwarových protokolů, se kterými pracuje koncový uživatel aplikace. Mezi takové technologie patří například HTTP pro přenos hypertextových dokumentů nebo technologie spojené s elektronickou poštou – SMTP, POP3 a IMAP (8).

1. 8 TCP/IP

TCP/IP je jedním z nejpopulárnějších, v praxi rozšířených, síťových modelů dnešní doby a bývá uplatňován ve velkém množství produktů a sítí. Jeho hlavní výhodou oproti modelu ISO/OSI je jeho praktická orientovanost, díky které na něm bylo postaveno mnoho významných standardů (8).

Na rozdíl od sedmivrstvého modelu ISO/OSI má TCP/IP pouze čtyři vrstvy, v nichž aplikační vrstva reprezentuje vrstvu aplikační, prezentační a relační, transportní vrstva víceméně odpovídá stejnojmenné vrstvě v ISO/OSI a vrstva síťového rozhraní zahrnuje vrstvu linkovou a fyzickou (8).

TCP/IP		ISO/OSI
Aplikační vrstva		Aplikační vrstva
		Prezentační vrstva
		Relační vrstva
Transportní vrstva		Transportní vrstva
Síťová vrstva		Síťová vrstva
Vrstva síťového rozhraní		Linková vrstva
		Fyzická vrstva

Obrázek č. 6 Porovnání vrstev TCP/IP a ISO/OSI
(Zdroj: 8)

Standardizované protokoly a technologie jsou významným prvkem komunikace na síti a většina současně používaných řešení vychází právě z rodiny TCP/IP, která se uchytila po celém světě. Na jejich fungování závisí služby, které používáme na denní bázi, jako jsou například webové aplikace, e-mailová komunikace a široké spektrum dalších technologií, které své fungování často staví na použití protokolu TCP či UDP (9).

1. 8. 1 Protokol IP

Internetový protokol IP (z anglického Internet Protocol) operuje na síťové vrstvě TCP/IP modelu a v současné době jsou v praxi využívány jeho dvě verze – IPv4 a IPv6. Zásadní

rozdíl mezi nimi spočívá v odlišné velikosti používaných IP adres, které slouží pro odlišení jednotlivých síťových uzlů (8).

V protokolu IPv4 jsou využívány 32 bitové adresy zapisované po bajtech v dekadické soustavě oddělených tečkou (například 192.168.0.1) a v protokolu IPv6 jsou uplatněny adresy o délce 128 bitů zapisované v hexadecimálním formátu po dvou bajtech (například 2001:0DB8:85B3:1234:4321:1278:1AC2:55AF). Právě větší adresní prostor byl hlavním důvodem vzniku protokolu IPv6, který však přináší i mnohé další změny a rozšíření, jako je vylepšení směrování, zavedení protokolu NDP pro objevování sousedů či vestavěné řešení kvality služeb (QoS) (8).

1. 8. 2 Protokol TCP

TCP (Transmission Control Protocol) je protokol vytvářející spojení mezi cílovými aplikacemi běžícími na koncových zařízeních a starající se o zajištění integrity přenášených dat pomocí kontrolních součtů. Jednotlivé aplikace spuštěné na koncových uzlech jsou odlišeny pomocí dvoubajtových čísel portů v rozsahu 0 až 65535, které mohou být jak standardizované, tak nestandardizované (9).

Mezi hojně používaná standardizovaná čísla portů patří například:

- 25 – SMTP (Simple Mail Transfer Protocol),
- 80 – HTTP (Hypertext Transfer Protocol),
- 143 – IMAP (Internet Message Access Protocol),
- 443 – HTTPS (Hypertext Transfer Protocol Secure) (8).

1. 8. 3 Protokol UDP

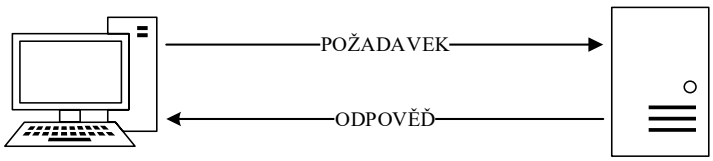
UDP (User Datagram Protocol) je komunikační protokol, který nezajišťuje spolehlivost a nevytváří mezi stranami žádné spojení. Jeho výhodou oproti TCP je možnost zasílat takzvané oběžníky, jež mohou mít více příjemců zároveň, čímž může být značně sníženo zatížení sítě, a protokol tak nachází využití například ve streamovacích službách (9).

Mezi protokoly podporující komunikaci na protokolu UDP patří například DNS (Domain Name System), jenž slouží k překladu IP adres na názvy ve formátu FQDN (Fully Qualified Domain Name), nebo protokol NTP (Network Time Protocol), který je využíván v počítačových sítích pro synchronizaci času (9).

1. 9 Protokoly HTTP a WebSockets

HTTP je protokol aplikační vrstvy používaný pro přenos informací mezi klientem a serverem v prostředí internetu od roku 1990. Princip HTTP je založen na zasílání požadavků od klienta na server a přijímání odpovědí ze serveru, přičemž požadavky i odpovědi mají standardizovaný formát (8).

Hlavní využití nachází protokol HTTP v oblasti webových technologií, u nichž dochází pomocí tohoto protokolu především k přenosu kódu a obsahu stránek, a webových služeb, u kterých je protokol použit například pro přenos zpráv protokolu SOAP (Simple Object Access Protocol) (8).



Obrázek č. 7 HTTP požadavek a odpověď
(Zdroj: Vlastní zpracování dle: 8)

Protokol WebSockets vznikl jako odezva na poptávku po možnosti obousměrné komunikace mezi klientem a serverem v prostředí webových aplikací. Dříve byl používán pouze protokol HTTP, který se pro mnohé řešení stává nevhodným kvůli svému striktnímu principu založeném na požadavku a odpovědi, v němž nemůže server zaslat klientovi data, aniž by nejdříve klient poslal požadavek (10).

Výhoda protokolu WebSockets je, že pro navázání spojení využívá protokolu HTTP, což mu umožňuje využít ke své funkci existujících infrastruktur a softwarových i hardwarových řešení (10).

Po navázání spojení pomocí zaslání požadavku a odpovědi, v nichž se klient domlouvá se serverem na použitých rozšířeních, verzích a protokolech, dochází k zasílání standardizovaných WebSocketových rámců, které již nepodléhají povinné HTTP struktuře. Díky tomu WebSockets zásadně snižuje přenosové režijní náklady (10).

1. 10 Penetrační testy

Penetrační testy (zkráceně pentesty) jsou testy spočívající ve vyhledávání bezpečnostních zranitelností v softwaru, počítačových sítích, IoT zařízeních a mnoha dalších technologických prvcích. Cílem penetračních testů je nalezení existujících chyb dříve, než by je nalezli a zneužili skuteční útočníci (11).

Zaměření penetračních testů může být různé, a tak se v praxi můžeme sekat s penetračními testy orientovanými kupříkladu na interní infrastrukturu, na webové i mobilní aplikace, na DNS servery či IoT zařízení. Nejen počítačový hardware a software je však zranitelnostmi ohrožen, a proto se provádí například také testy zaměřené na lidský faktor pomocí metod sociálního inženýrství (11).

Pro penetrační testy vznikl standard zvaný PTES (Penetration Testing Execution Standard), který nezávazně definuje procedury na základě léty prověřených postupů a usnadňuje firmám provádět kvalitní penetrační testy, a to od počátečního ujednání zadání testů až po odevzdání profesionální finální zprávy (12).

Samotné penetrační testy dělíme dle standardu PTES na následující fáze:

1. Počáteční ustanovení,
2. Získávání informací,
3. Modelování hrozeb,
4. Analýza zranitelností,
5. Exploitace,
6. Fáze po exploitaci,
7. Reportování (12).

1. 10. 1 Počáteční ustanovení

Počáteční ustanovení jsou fází, při níž je nutné si s klienty dohodnout základní parametry penetračního testování, aby nedošlo k nechtěným potížím při spuštění testů. Jde především o definování cílů testování, typu prováděných testů, projednání formátu výsledných zpráv, domluvy platebních termínů, vymezení časového okna pro testování a další činnosti spojené s administrativou nutnou pro bezproblémové spuštění testů (12).

1. 10. 2 Získávání informací

Získávání informací spočívá ve shromažďování všech informací o cílovém systému i jeho uživatelích za účelem vytvoření informační báze pro samotný test. K této fázi mohou být použity například internetové vyhledávače, skenery portů, účetní závěrky, profily zaměstnanců na sociálních sítích a další dostupná data (12).

1. 10. 3 Modelování hrozeb

Modelování hrozeb je aktivitou, při níž testeři identifikují hrozby před samotným testem, aby se mohli zaměřit na nejkritičtější části systému. Pro tyto účely jsou využívány poznatky z druhé fáze a znalosti fungování detekovaných subsystémů, jimiž tester disponuje (12).

1. 10. 4 Analýza zranitelností

Analyzování zranitelností je zaměřena na nacházení slabých míst jednotlivých subsystémů na základě automatizovaných i ručních testů, které mohou být založeny jak na detekci verzí použitého software, tak na různých technikách lišících se v závislosti na druhu subsystému, jako jsou například:

- použitá výchozí hesla,
- skryté stránky ve webových aplikacích,
- eskalace privilegií,
- přetečení bufferu,
- nešifrovaný přenos citlivých dat (12).

1. 10. 5 Exploitace

Exploitace (zneužití) nalezených zranitelností bývá založena na použití specializovaných nástrojů i ručním testování, při kterých se testeři pokouší obejít zabezpečení a dostat se do cílového systému bez povšimnutí (12).

1. 10. 6 Fáze po exploitaci

Po úspěšné exploitaci se penetrační tester snaží o získání informací o dané službě a pokouší se zneužít získané přístupy a data pro další postup v útoku stejně jako by postupoval reálný útočník. Rozdíl je však v tom, že se oproti útočníkovi snaží na cílovém systému nezpůsobit žádné škody a jeho cílem je zjistit, jak závažný problém by úspěšná exploitace způsobila v případě napadení skutečným útočníkem (12).

1. 10. 7 Reportování

V poslední fázi penetrační testeři vyhotovují zprávu obsahující popisy jednotlivých nálezů včetně ohodnocení jejich závažnosti a doporučení pro zabezpečení. Výsledná zpráva často obsahuje dvě části obsahující popisy chyb o rozdílných rozsazích, a to část

pro management, v níž jsou sepsány shrnutí a rizika jednotlivých zranitelností, a část technickou, která je zaměřena na podrobné popisy nálezů pro jejich správnou opravu (12).

1. 11 Proxy

Proxy server je prostředník v síťové komunikaci mezi klientem a cílovým serverem, který vůči klientovi vystupuje jako cílový server a vůči serveru jako zdrojový klient. Výhodou proxy je, že mimo pouhého předávání dat ze strany na stranu umožňuje také manipulaci dat, filtraci a cachování (9).

V penetračních testech jsou proxy velmi významným nástrojem pro testování aplikací, používaným zejména pro aplikace pracující na protokolu HTTP, u nějž umožňují zachycovat a modifikovat požadavky klienta a odpovědi serveru. Klientské aplikace mnohdy ošetřují vstupy od uživatele a posílají je v již bezpečném formátu. Pokud však dojde k modifikaci dat na cestě mezi klientem a serverem, může dojít k obejití autorizace, provádění příkazů nebo například získání citlivých dat (13).

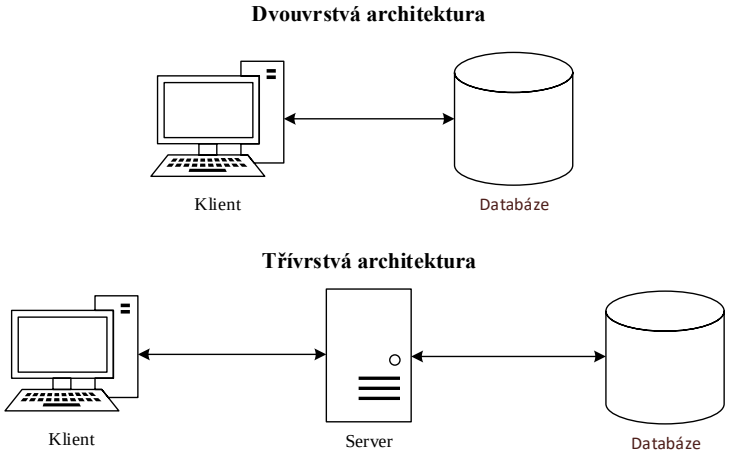
1. 12 SSL/TLS

Protokoly SSL a TLS jsou sady kryptografických protokolů, které slouží k bezpečnému přenosu dat pomocí šifrování dat na transportní vrstvě. Protokol SSL (z anglického Secure Sockets Layer) je starší protokol, který je v současné době nahrazován novějším protokolem TLS (Transport Layer Security) (8).

Protokoly SSL/TLS jsou jedny z nejrozšířenějších kryptografických protokolů, které v sobě zahrnují nespočet různých algoritmů, a tak se s touto sadou můžeme v praxi setkat u rozsáhlého spektra technologií jako jsou HTTP, FTP či jednotlivé protokoly elektronické pošty (8).

1. 13 Penetrační testování tlustých klientů

Tlustí klienti je označení pro síťové počítačové aplikace zakládající se na dvouvrstvé nebo třívrstvé architektuře, které poskytují uživateli rozsáhlou funkcionalitu. Tento typ aplikací vyžaduje speciální přístup při penetračním testování, jelikož se od populárních webových tenkých klientů značně liší, a to zejména rozdílnými technologiemi a větším objemem dat zpracovávaných na straně klienta (14).



Obrázek č. 8 Dvouvrstvá a třívrstvá architektura
(Zdroj: Vlastní zpracování dle: 14)

Během testování tlustých klientů je nejzákladnější fází získávání informací o fungování aplikace, kterými jsou například použité softwarové technologie, využívané síťové protokoly, adresy cílových serverů nebo seznamy lokálních souborů využívaných pro čtení či zápis. Po získání potřebných informací se penetrační testeři zabývají samotným testováním, při němž se snaží zanalyzovat a narušit síťovou komunikaci, nalézat citlivé údaje pomocí reverzního inženýrství a nacházet a zneužívat další bezpečnostní zranitelnosti (14).

2 ANALÝZA PROBLÉMU A SOUČASNÉ SITUACE

V současné době se při testování tlustých klientů bezpečnostní experti zabývají především aplikacemi využívajícími protokol TCP, respektive aplikační protokoly postavené nad protokolem TCP, jako jsou HTTP, WebSockets, FTP a mnohé další standardizované i nestandardizované protokoly. Pro testy těchto síťových aplikací používají testeři různé softwarové nástroje, jež umožňují odposlouchávání komunikace nebo modifikaci přenášených dat.

Následující kapitoly využiji k analýze současné situace v oboru penetračních testů tlustých klientů a rozboru existujících řešení, která jsou bezpečnostními experty v současnosti využívána. Pomocí dat získaných jednotlivými analýzami následně navrhnu požadavky pro vývoj vlastního softwarového řešení.

2. 1 Současná situace testování tlustých klientů

Poslední dobou roste ze strany velkých i malých společností zájem o ověření bezpečnosti jimi používaných nebo vyvíjených aplikací, a to jak z důvodu nárůstu hackerských útoků, tak z legislativních důvodů. Mnohá právní nařízení dávají společnostem povinnost nechat prověřovat bezpečnost svých aplikací či síťových infrastruktur, což vede k nárůstu poptávky po rozličných penetračních testech.

Po několik let byla vysoká poptávka především po penetračních testech webových aplikací, avšak postupně si začínají společnosti uvědomovat citlivost dat, která zpracovávají prostřednictvím tlustých klientů, a tak roste zájem i o jejich testování. Silná poptávka po bezpečnostních testech webových aplikací bohužel podpořila a stále podporuje zejména vývoj v oblasti nástrojů pro protokol HTTP a kvalita nástrojů pro samotný protokol TCP, hojně využívaný v tlustých klientech, narůstá velmi pomalým tempem.

Při testech tlustých klientů se používají různé veřejné i interní metodiky, které spočívají v analýzách zdrojů využívaných testovanou aplikací, analýzách síťové komunikace a analýzách ve formě reverzního inženýrství, při nichž dochází k odhalování interního fungování aplikace a pochopení probíhající síťové komunikace, ve snaze najít způsoby k obejití bezpečnostních restrikcí. Hackeři se u tlustých klientů zaměřují na kryptografické slabiny, chyby v autentizačním a autorizačním procesu, nesprávnou validaci vstupů, chyby v aplikační logice a mnoho dalších nedostatků, jež mohou vést

k porušení alespoň jednoho ze tří základních atributů bezpečnosti (integrity, dostupnosti či důvěrnosti).

K vyhledávání bezpečnostních zranitelností používají různé nástroje jako jsou dekompilátory, sniffery, systémové utility, monitory procesů nebo nástroje pro modifikaci grafického rozhraní. Jednotlivé nástroje používané pro testování datových přenosů aplikací, jež jsou předmětem této bakalářské práce, rozeberu v kapitole analýz konkurenčních aplikací.

2. 2 Používané operační systémy

Při vývoji aplikací je nutné zvážit, pro jaký operační systém budou výsledné aplikace určeny, aby mohly být vybrány vhodné technologie. V následujících dvou podkapitolách rozeberu analýzu používaných operačních systémů, která poslouží jako podklad pro výběr cílové platformy pro nástroj navrhovaný a vyvíjený v rámci mé bakalářské práce.

Vzhledem k tomu, že penetrační testování je velmi složitý proces vyžadující provádění mnoha různých činností, které je z hlediska efektivity nejlepší provádět na stolním počítači či notebooku, nejsou většinou pro ruční testování používány mobilní zařízení. V případě, že je nutné testovat aplikaci pro mobilní zařízení, bývá snaha provádět velkou část testů vzdáleně. Z těchto důvodů se v následující analýze zaměřuji pouze na desktopové operační systémy, pro něž má smysl uvažovat podporu navrhovaného softwarového nástroje.

2. 2. 1 Operační systémy penetračních testerů

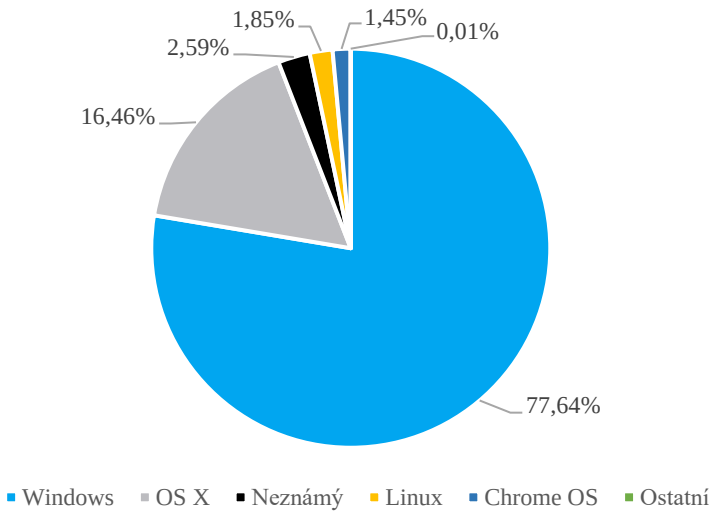
Penetrační testeři využívají ke své práci různé operační systémy a jejich volba se mnohdy liší v závislosti na osobních preferencích a konkrétním cíli testů. Mezi používané operační systémy tak mohou patřit například rozličné verze Microsoft Windows i široké spektrum linuxových distribucí.

Linuxové distribuce jsou u hackerů velmi oblíbené především pro svoji otevřenost a rozsáhlejší možnosti po stránce nastavení systému. Existuje zároveň mnoho linuxových distribucí, které jsou už od samého počátku přizpůsobeny pro penetrační testování a obsahují odladěné, k použití připravené, spektrum populárních nástrojů pro vyhledávání a zneužívání bezpečnostních zranitelností. Mezi populární distribuce patří například Kali Linux, Samurai Web Testing Framework či Parrot Security OS (15).

Za tvorbou hackerských distribucí stojí mnohdy velká komunita bezpečnostních expertů, kteří pomáhají sestavit optimální sadu nástrojů integrovaných v jednom systému. Jednotlivé distribuce se liší zejména v použitém prostředí, náročnosti na hardware a poskytovaných nástrojích – například distribuce Network Security Toolkit obsahuje nástroje užitečné pro testování počítačových sítí (15).

2. 2. 2 Operační systémy testovaných aplikací

Jelikož je cílem práce navržení a vytvoření softwarového nástroje pro testování aplikací používajících TCP protokoly, je nutné zvážit i závislost testovaných aplikací na operačním systému a zvolit vhodný kompromis, aby nemuseli penetrační testeři při práci využívat většinu času virtualizační software.



Graf č. 1 Podíl desktopových operačních systémů na celosvětovém trhu v prosinci 2019
(Zdroj: Vlastní zpracování dle: 16)

Statistická data o podílu desktopových operačních systémů na celosvětovém trhu od společnosti StatCounter udávají, že většinové zastoupení má Microsoft Windows (77,64 %), a to s velkým náskokem oproti druhému nejpoužívanějšímu desktopovému operačnímu systému, kterým je OS X od společnosti Apple (16,46 %). Zbylých 5,9 % trhu se dělí mezi Linux, Chrome OS a další, u koncových uživatelů méně populární, operační systémy (16).

2. 2. 3 Shrnutí operačních systémů

Z výše provedených analýz jsem vyvodil, že většina testovaných aplikací bude s největší pravděpodobností naprogramována s podporou zejména operačního systému Microsoft

Windows a OS X, které mají dohromady 94,1% podíl na trhu desktopů. Oproti tomu penetrační testeři budou častokrát preferovat rozličné distribuce postavené na Linuxovém jádře, které jsou pro provádění bezpečnostních testů přizpůsobeny.

Navrhovaná a vyvíjená aplikace by tak měla v ideálním případě podporovat jak operační systémy Windows a OS X (populární u uživatelů), tak Linuxové distribuce (populární u pentesterů), aby neomezovala možnosti etických hackerů.

2. 3 Konkurenční aplikace

Testování tlustých klientů je proces, k jehož kvalitnímu provádění se v praxi používají různé softwarové nástroje, které bezpečnostním specialistům pomáhají vyhledávat zranitelnosti pomocí nahlížení pod běžně viditelný povrch aplikací. Ve své bakalářské práci jsem se rozhodl zaměřit na software pro práci se síťovou komunikací počítačových aplikací a předsevzal jsem si navrhnout a vyvinout nástroj, který napraví nedostatky existujících řešení, a proto se v následujících kapitolách budu věnovat analýze nejznámějších nástrojů používaných penetračními testery k analýze a modifikaci síťového přenosu tlustých klientů.

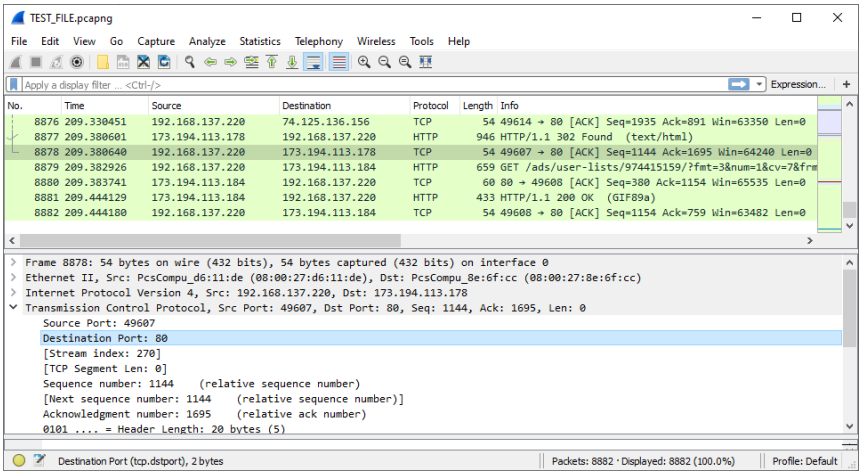
2. 3. 1 Wireshark

Wireshark je open-source nástroj pro analýzu síťových paketů, dostupný pro operační systém Microsoft Windows i unixové operační systémy, a patří v současnosti mezi nejpoužívanější software svého druhu. Z hlediska licence Wireshark spadá pod otevřenou licenci GNU General Public Licence verze 2 (GPLv2), a lze jej tak používat bez placení jakýchkoliv poplatků i v komerčním prostředí (17).

Tento nástroj je dnes hojně využíván pro analýzu komunikace tlustých i tenkých klientů především díky nespočtu podporovaných formátů, protokolů (DNS, HTTP, ICMP, …) a kódování (ASCII, EBCDIC, UTF-8, …). Možnosti aplikace jsou navíc i nadále rozvíjeny jak formou aktualizací aplikace, tak vznikem nových rozšíření, která je možné do aplikace vyvíjet například pomocí skriptovacího jazyku Lua.

Wireshark se řadí do kategorie takzvaných snifferů, které staví na odposlouchávání komunikace a neumožňují provádění zásahu do přenášených dat, což je jejich největší nevýhodou při provádění penetračních testů, při nichž je velmi často nutné provést určité modifikace síťového provozu, které způsobí narušení bezpečnosti testované aplikace.

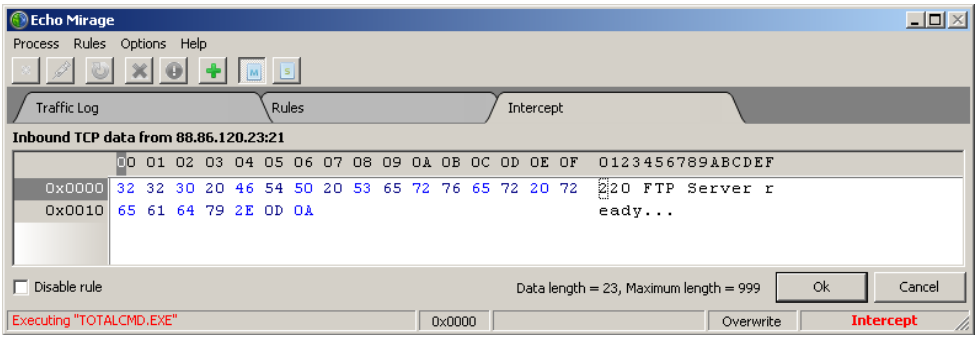
Naproti tomu má Wireshark velkou výhodu založenou na podpoře více operačních systémů a pro svoji komplexnost jde o velmi kvalitní nástroj pro analytickou část testů. Princip jeho fungování navíc nevyžaduje provádění žádných speciálních operací s testovanou aplikací, čímž se liší například od proxy řešení, u nichž je potřeba zařídit, aby aplikace komunikovala přes separátní proxy server.



Obrázek č. 9 Snímek z aplikace Wireshark
(Zdroj: Vlastní zpracování dle: 17)

2. 3. 2 Echo Mirage

Echo Mirage je jednou z nejpoužívanějších bezplatných aplikací pro testování síťové komunikace tlustých klientů. Je založena na takzvaném hookování procesů, které se opírá o funkce operačního systému, jež poskytují API pro napojení na knihovny používané testovanou aplikací. Na tyto knihovny se Echo Mirage napojí a poskytne bezpečnostnímu specialistovi grafické prostředí pro modifikaci probíhající komunikace (14).



Obrázek č. 10 Snímek z aplikace Echo Mirage
(Zdroj: Vlastní zpracování dle: 18)

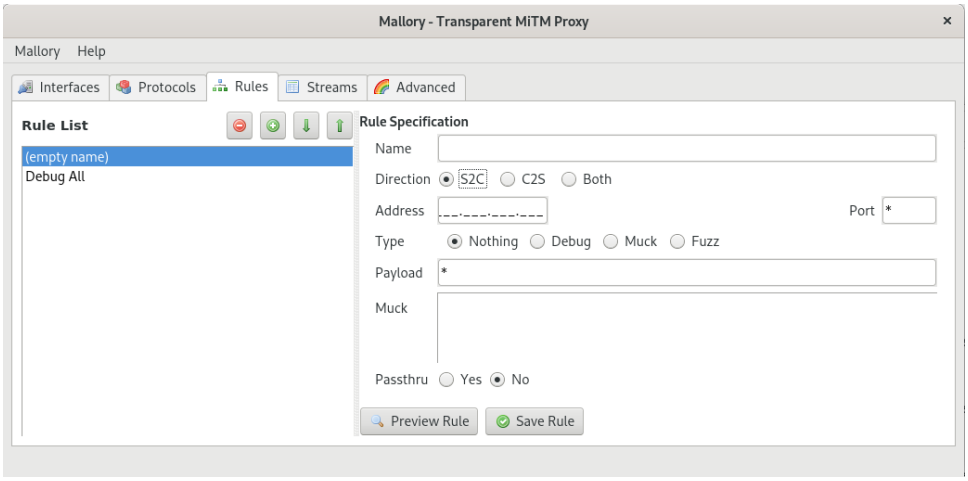
Nástroj Echo Mirage dostal svoji poslední aktualizaci v roce 2013 ve formě verze 3.1, kterou lze vidět na výše uvedeném snímku. Dnes již není vyvíjen a webové stránky vývojářů nejsou dostupné.

Velkou nevýhodou aplikace Echo Mirage a ostatních aplikací využívajících hookování procesů je jejich závislost na konkrétním operačním systému a jeho funkcích, kvůli čemuž je nástroj Echo Mirage dostupný pouze pro operační systém Microsoft Windows. Princip hookování je navíc také závislý na knihovnách použitých aplikací a nástroj musí mít naimplementovanou podporu pro tyto knihovny, jinak není možné se na komunikaci testované aplikace napojit. Další značnou nevýhodou programu Echo Mirage, a to zejména v kombinaci s jeho ukončeným vývojem, je absence rozšiřitelnosti, díky které by bylo možné implementovat chybějící funkcionalitu.

Výhodou procesu hookování je jeho transparentnost vůči testované aplikaci, která nemusí být žádným způsobem přizpůsobována, aby mohly být prováděny její bezpečnostní testy. Jediný požadavek je, aby aplikace používala pro síťovou komunikaci či šifrování knihovny, které jsou podporovány hookovacím nástrojem.

2. 3. 3 Mallory

Mallory je nástroj napsaný v jazyce Python, který umožňuje vytvářet MITM proxy běžící ve formě síťové brány. Jde o jeden z nepoužívanějších nástrojů pro TCP a UDP protokoly, který je ale bohužel už několik let od svého vzniku neudržovaný a nadále nevyvíjený (19).



Obrázek 1 Snímek z aplikace Mallory
(Zdroj: Vlastní zpracování dle: 19)

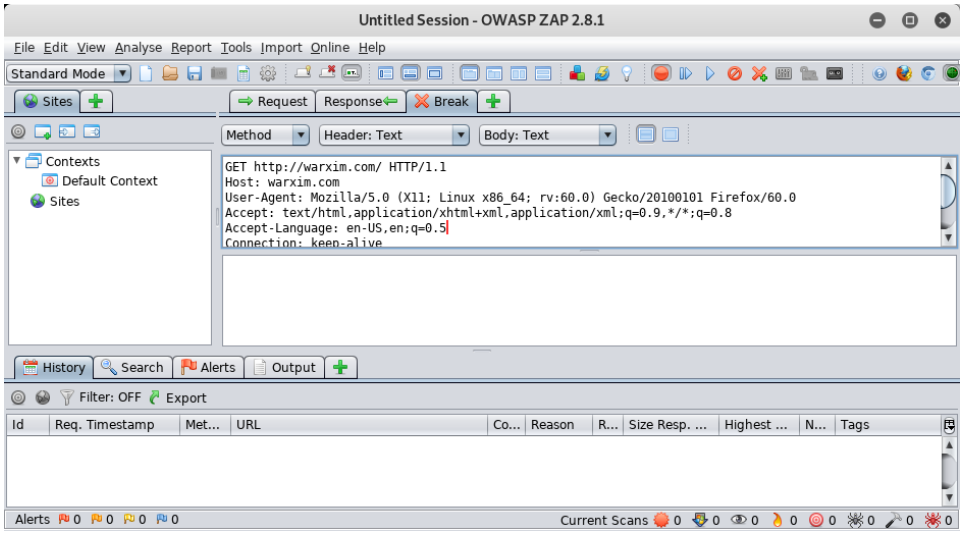
Do dnešního dne pro nástroj Mallory neexistuje žádná forma dokumentace pro uživatele ani pro vývojáře rozšíření. V případě zájmu o přidání vlastní funkcionality musí externí vývojáři nejdříve důkladně prozkoumat zdrojové kódy celé aplikace, aby byli schopni naprogramovat nové rozšíření.

Velkou výhodou aplikace Mallory je její kompatibilita s většinou používaných operačních systémů a jednoduché grafické rozhraní, v němž jde operovat s daty přenášenými testovanou aplikací.

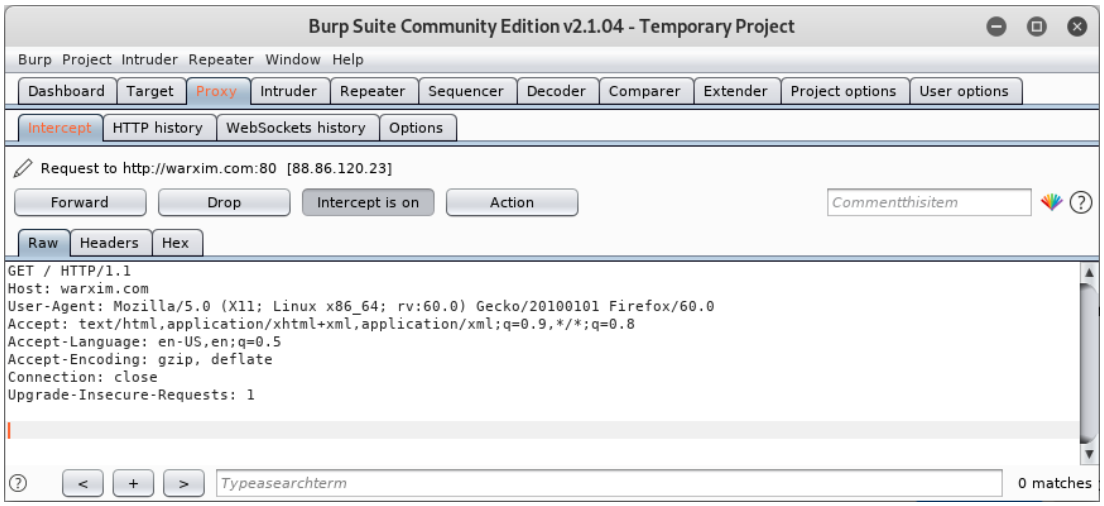
2. 3. 4 ZAP Proxy a Burp Suite

Dalším populárním nástrojem pro testování komunikace aplikací jsou proxy, které se vydávají na straně serveru za klienta a na straně klienta za server, díky čemuž mají naprostou kontrolu nad řízením probíhající síťové komunikace.

Mezi nejpoužívanější proxy aplikace pro penetrační testy se řadí komerční nástroj Burp Suite a open-source aplikace ZAP (Zed Attack Proxy) spadající pod OWASP (Open Web Application Security Project). Jak ZAP, tak Burp Suite jsou navrženy pouze pro protokoly používané v prostředí webových aplikací (HTTP a WebSocket) a nejsou přizpůsobeny pro testování tlustých klientů používajících jiné síťové protokoly.



Obrázek č. 11 Snímek z aplikace ZAP Proxy
(Zdroj: Vlastní zpracování dle: 20)



Obrázek č. 12 Snímek z aplikace Burp Suite
(Zdroj: Vlastní zpracování dle: 21)

Pro protokol TCP existuje velmi málo kvalitních hackerských proxy, a penetrační testeři tak mají velmi úzké výběrové spektrum softwarových řešení. Mezi současná proxy řešení patří například rozšíření do výše uvedeného komerčního programu Burp Suite zvané NoPE (Burp-Non-HTTP-Extension), jenž dostalo poslední aktualizaci před 3 lety a dodnes neumožňuje využít naplno kvalit nástroje, v jehož prostředí funguje.

O využití existujících HTTP proxy řešení pro jiné protokoly se snaží také open-source nástroj mitm_relay dostupný na platformě GitHub, který však podporuje pouze základní editaci komunikace a nelze pomocí něj provádět žádnou formu opakování požadavků, jelikož to nedovoluje jeho architektura.

Nejpodstatnější výhodou proxy pro testování je absolutní moc nad probíhající komunikací spočívající v prakticky neomezených možnostech při její modifikaci a správě.

Nevýhodou proxy řešení je, že v některých situacích může být jejich použití problematičtější, a to například v případě, kdy testovaná aplikace využívá ochranných mechanismů bránících MITM útokům nebo obsahuje napevno zapsanou IP adresu cílového serveru.

2. 3. 5 Shrnutí konkurenčních aplikací

V současné situaci se na trhu aplikací pro penetrační testy síťové komunikace vyskytuje mnoho různých produktů, které nepokrývají zcela požadavky na kvalitní testování

síťových aplikací využívajících protokol TCP a další na něm stavějící protokoly. Mnohá softwarová řešení jsou navíc několik let nevyvíjená a jejich budoucnost je velmi nejistá.

2. 4 Vyhodnocení analýz a vymezení požadavků na aplikaci

Provedené analýzy ukázaly, že současná situace nabízí penetračním testerům široké spektrum softwarových řešení, která však mají mnoho nedostatků a často nejsou vhodné pro testování tlustých klientů používajících jiné než webové protokoly. Právě velké množství nedostatků existujících aplikací a zvyšující se poptávka po penetračních testech dává prostor pro vznik nového nástroje, který bude cílit na bezpečnostní specialisty zabývající se penetračními testy aplikací, jež používají síťovou komunikaci založenou na protokolu TCP a protokolech z něj vycházejících.

Na základě výhod a nevýhod existujících řešení a na základě vlastních zkušeností s použitím uvedených softwarových řešení v praxi jsem sestavil požadavky, které bude nutné implementovat do navrhované aplikace. Jednotlivé požadavky jsou podrobně rozepsány v úvodu následující kapitoly.

Pro textaci nástroje jsem zvolil anglický jazyk, jelikož jde o nejpoužívanější mezinárodní jazyk v oblasti informačních technologií a je v něm lokalizována většina softwarových řešení využívaných pro penetrační testování (včetně všech analyzovaných konkurenčních aplikací).

3 VLASTNÍ NÁVRH ŘEŠENÍ, PŘÍNOS PRÁCE

Na základě analýz provedených v předcházející kapitole jsem se rozhodl začít s návrhem vlastního řešení, které se bude opírat o teoretická východiska z první kapitoly. Cílem mé bakalářské práce je navrhnout a vytvořit proxy nástroj jako funkční alternativu pro existující nástroje, která bude vhodná pro provádění penetračních testů aplikací využívajících aplikační protokoly stavějící na protokolu TCP pro síťovou komunikaci.

Při návrhu se nejdříve zaměřím na hlavní vlastnosti, které vytyčuji na základě předešlých analýz, provedu návrh jádra celé aplikace a rozepíši postupy a návrhy použité při tvorbě jeho jednotlivých částí. Poté se budu zabývat vývojem jednotlivých rozšíření, která do vytvářeného systému zavedou hlavní požadovanou funkcionalitu.

Vzhledem k účelu aplikace jsem se rozhodl zvolit název PETEP, jenž je zkratkou pro PEnetration TEsting Proxy.

3. 1 Hlavní vlastnosti řešení

Na základě teoretických východisek, analýzy současného stavu a několikaleté praxe v oboru, jsem definoval nejdůležitější vlastnosti, které musí mnou navrhované řešení splňovat:

- modularita,
- rozšiřitelnost,
- multiplatformnost,
- grafické rozhraní,
- podpora více paralelně otevřených spojení a proxy.

3. 1. 1 Modularita

Modularita vytvářené aplikace bude spočívat v rozvržení jednotlivých funkcí do samostatných modulů, které bude možné na základě projektových požadavků přidávat, odebírat a spravovat uživatelem pomocí grafické konfigurace.

Nejzásadnější výhodou modularity je přizpůsobitelnost nástroje pro konkrétní zadání testu a možnost šetřit s výpočetními zdroji – pamětí a procesorem. Takové úspory výpočetních zdrojů pak mohou vést k podstatně rychlejšímu vykonávání penetračních testů díky zjednodušení, optimalizaci prostředí a značným úsporám finančním.

3. 1. 2 Rozšiřitelnost

Rozšiřitelnost aplikace si musí zakládat na možnosti třetích stran programovat vlastní moduly, což umožní penetračním testerům zavádění nové funkcionality, kterou mohou během práce používat, aniž by museli zasahovat přímo do hlavního aplikačního kódu. Z dlouhodobého hlediska rozšiřitelnost aplikace navíc značně usnadní a urychlí přizpůsobitelnost aplikace neustálým změnám požadavků uživatelů, k nimž dochází vlivem vzniku nových technologií.

Celá rozšiřitelnost aplikace se bude opírat o načítání externích JAR balíčků, které obsahují kód využívající jednotné API distribuované společně s jednotlivými vydáními aplikace.

3. 1. 3 Multiplatformnost

Z provedených analýz vyplývá, že mnohé nástroje postrádají zásadní vlastnost, kterou je nezávislost na operačním systému. Přitom penetrační testování tlustých klientů je nutné mnohdy provádět nejen na systémech Microsoft Windows, ale například také na unixových operačních systémech jako je Linux a OS X. Linuxové distribuce jsou navíc mezi bezpečnostními experty často preferovány pro své vlastnosti.

Pro multiplatformní funkcionalitu jsem zvolil programovací jazyk Java, který umožní chod aplikace na desktopových operačních systémech, aniž by bylo nutné během vývoje řešit podporu jednotlivých platforem zvlášť, a přitom poskytne solidní výkon i široké spektrum knihoven pro usnadnění vývoje.

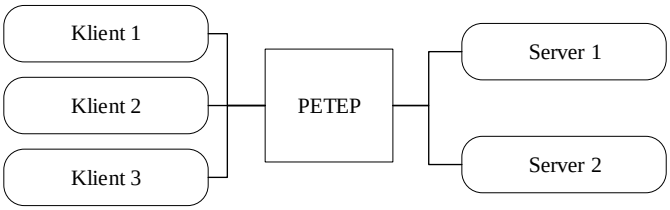
3. 1. 4 Grafické rozhraní

Penetrační testeři jsou při práci zvyklí pracovat i s konzolovými nástroji, avšak pro analýzu a editaci datové komunikace jsou textové aplikace velmi nepohodlné a značně ztěžují provádění testů. Grafické rozhraní je tak zcela nutnou podmínkou pro úspěch aplikace a vzhledem k předcházejícím požadavkům si musí umět poradit s výše zmíněnou modularitou a rozšiřitelností.

Pro vytváření grafického rozhraní jsem použil technologii JavaFX rozebranou v teoretických východiscích práce, která vychází vstříc všem základním požadavkům na aplikaci. Zároveň podporuje moderní a efektivní způsob návrhu grafického prostředí pomocí jeho oddělení od zdrojového kódu aplikace a uložení v samostatných šablonách ve formátu FXML.

3. 1. 5 Více paralelně otevřených spojení a proxy

Vzhledem k tomu, že reálné aplikace používají v některých případech více paralelně otevřených spojení na jednu či více cílových služeb, rozhodl jsem se přidat ke své práci požadavek na to, aby nástroj podporoval více paralelně otevřených spojení i více současně běžících proxy serverů v rámci jedné instance.



Obrázek č. 13 Více klientů a více serverů v jedné instanci
(Zdroj: Vlastní zpracování)

V případě potřeby tak bude možné pracovat s více otevřenými spojeními jednodušším způsobem a aplikovat na jednotlivá spojení a služby totožná modifikační a jiná pravidla.

3. 2 Návrh jádra

V této kapitole rozeberu návrh jednotlivých částí jádra a návrh částí na něj přímo navazujících. Ty samy o sobě fungují jako základ pro propojení a zprovoznění jednotlivých rozšíření a modulů a umožňují aplikaci fungovat nezávisle na konkrétních implementovaných protokolech a procesech pro zpracování dat.

3. 2. 1 Protokolová datová jednotka

Pro abstrakci zasílaných zpráv v celé aplikaci je použita třída PDU (z anglického Protocol Data Unit), která stanovuje základní vlastnosti datových balíčků posílaných od klienta na server a ze serveru ke klientovi prostřednictvím proxy. Konkrétní implementace datových jednotek pak záleží na vývojáři interních či externích proxy modulů, které zahajují jednotlivá spojení, v nichž dochází k vytváření datových jednotek specifických pro daný protokol. Obsah datových balíčků lze kategorizovat do dvou částí, a to na část popisující tok dat a datovou část.

Popisná část PDU balíčků obsahuje informace o zdrojové proxy, spojení a směru, které jsou důležité pro směrování PDU napříč aplikací. Dále obsahuje množinu řetězců sloužících k označování balíčku takzvanými tagy.

Datová část PDU je v základní třídě deklarována pouze ve formě abstraktních metod, které musí programátor interního či externího modulu protokolu naimplementovat.

Základní podmínkou je, že data musí být nastavitelná a získatelná prostřednictvím metod zobrazených na následujícím snímku úseku kódu třídy PDU.

```
/** Returns buffer. */
public abstract byte[] getBuffer();

/** Returns size of data in the buffer. */
public abstract int getSize();

/** Returns the buffer and size of data in the buffer. */
public abstract void setBuffer(byte[] buffer, int size);

/** Resizes the buffer. */
public abstract void resize(int size);
```

Obrázek č. 14 Abstraktní metody pro datovou část PDU
(Zdroj: Vlastní zpracování)

Jednotlivým implementacím PDU je mimo metod pro datovou část povinné nadefinovat funkcionalitu pro vytváření kopie jednotky a metody pro získání a nastavení znakové sady, díky nimž je možné snadno převádět datové bajty do textové reprezentace.

Součástí API je mimo abstraktní třídy PDU také její základní implementace ve formě třídy DefaultPdu, která úložiště dat realizuje na bázi jednoduchého bufferu vytvořeného pomocí pole bajtů a numerické hodnoty pro velikost dat.

Datové jednotky je nutné v aplikaci ukládat do front, aby si přes ně mohly jednotlivé moduly a jádro navzájem předávat PDU. Pro tyto účely vznikla třída PduQueue, která obstarává frontu datových jednotek zabezpečenou proti současným přístupům z více vláken.

3. 2. 2 Spojení

Spojení je v aplikaci realizováno pomocí abstraktní třídy Connection, jejíž konkrétní implementace je závislá na modulu proxy, který poskytuje řešení pro konkrétní komunikační protokol. Hlavním účelem spojení je přijímání a odesílání balíčků PDU na obou stranách proxy (na straně mezi klientem a proxy a mezi proxy a serverem).

Základními proměnnými uvnitř spojení jsou fronty pro odchozí balíčky v obou směrech, do nichž se při běhu aplikace přidávají jednotlivá PDU, a číselný identifikátor spojení, který je používán napříč celým nástrojem pro jednoznačnou identifikaci spojení v rámci jedné instance proxy. Jedním ze zásadních účelů, pro které vznikl jednoznačný identifikátor spojení, je jeho využitelnost při serializaci a deserializaci balíčků na textové řetězce, při nichž je číselný identifikátor použit jako zastupující hodnota pro celý objekt spojení.

Pro konkrétní implementaci spojení je nutné definovat zejména metody pro spuštění a ukončení spojení, v nichž by mělo docházet například ke spuštění vláken pro čtení a zápis do příslušných soketů na jednotlivých stranách spojení. V případě, že by do nástroje měl být implementován protokol, který nenavazuje spojení, může být třída použita jako uložiště pro popisná data o koncových stanicích komunikace, aby bylo možné opakovat jednotlivé zasílané požadavky.

Správa jednotlivých spojení, která mohou být paralelně aktivní, probíhá prostřednictvím definice potomka třídy ConnectionManager, která udává základní metody pro práci se spojeními, jež mohou být využívány napříč celou aplikací.

```
11 @PetepAPI
12 public abstract class ConnectionManager {
13   /** Returns connection by ID. */
14   public abstract Connection get(int id);
15
16   /** Adds connection to the connection manager. */
17   public abstract boolean add(Connection connection);
18
19   /** Removes connection from the connection manager. */
20   public abstract boolean remove(Connection connection);
21
22   /** Removes connection from the connection manager. */
23   public abstract Connection remove(int id);
24
25   /** Returns list of connections. */
26   public abstract List<Connection> getList();
27
28   /** Stops all connections. */
29   public abstract void stop();
30 }
```

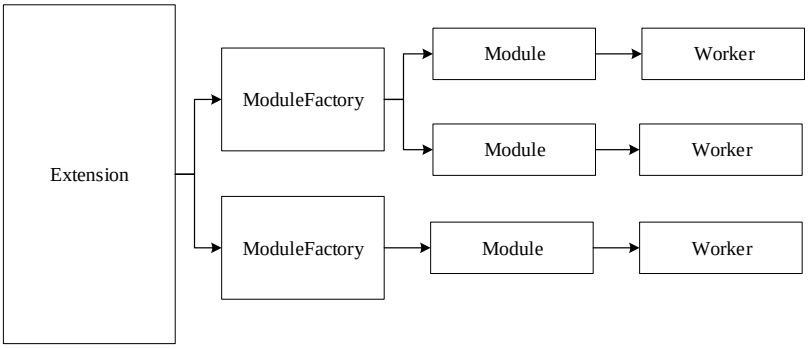
Obrázek č. 15 Třída ConnectionManager pro správu spojení
(Zdroj: Vlastní zpracování)

Součástí aplikačního API je základní implementace DefaultConnectionManager, jež poskytuje stabilní nástroj pro správu spojení založený na takzvané souběžné mapě hashů, která zabezpečuje metody proti chybám vznikajícím současným přístupem z více vláken. Klíčem v použité mapě je jednoznačný číselný identifikátor spojení a hodnotou je příslušná instance spojení.

3. 2. 3 Moduly

Moduly jsou zásuvné části, které jsou přidávány do aplikace interně či externě prostřednictvím rozšíření a komunikují s jádrem aplikace, jež je připraveno na použití modulů pro proxy a pro interceptory. Oba typy modulů mají společný programový základ, který je používán pro jejich ovládání a nastavování. Tento funkční základ

nejvýstižněji popisuje následující schéma zachycující možnou hierarchii jednotlivých částí modulů vycházejících z rozšíření.



Obrázek č. 16 Schéma modulů
(Zdroj: Vlastní zpracování)

Rozšíření zastoupené třídou Extension registruje do aplikace továrny pro moduly (ModuleFactory), které rozšiřují možnosti uživatele o vytváření nových typů modulů (Module) a umožňují jejich konfiguraci. Vytvořené moduly pak při startu jádra vytvoří na základě nastavení konkrétní instanci „pracovníka“ (Worker), jež je přiřazena na určité místo dosažitelné přímo z jádra aplikace.

Každá továrna modulů je jednoznačně identifikována unikátním řetězcem zvaným kód, pomocí kterého je možné provádět serializaci a deserializaci jednotlivých modulů při načítání a ukládání konfigurace projektu. Stejným způsobem jsou označeny i podřízené moduly, jimž jejich unikátní kód přidělují uživatelé aplikace při jejich definici. Unikátní kódy mohou být navíc velmi snadno využívány pro komunikaci mezi odlišnými moduly a v případě modulů proxy je na kódech postavena například i serializace instancí proxy do HTTP formátu, aby bylo při následné deserializaci možné určit, na jakou proxy má být sestavená datová jednotka poslána.

3. 2. 3. 1 Moduly pro proxy

Moduly pro proxy do aplikace přidávají nejdůležitější a neopomenutelný prvek, kterým je zpracování konkrétního protokolu a kompletní obsluha komunikace mezi klientem, proxy a serverem. Jejich prací je zejména obsluha následujících tří procesů:

- spuštění a ukončení proxy,
- správa spojení (zahajování, udržování a ukončování),
- serializace a deserializace datových jednotek.

Jádro nástroje je přizpůsobeno pro současnou činnost více spojení i proxy, kdy proxy navzájem sdílí jednu soustavu modulů pro zpracování dat. Tato vlastnost podporuje nastavitelnost společných pravidel pro modifikaci a označování jednotlivých datových jednotek přicházejících z rozdílných spojení či služeb, a zároveň usnadňuje pozastavování komunikace v případech, kdy je ke komunikaci použito více paralelních spojení.

Volnost ve vývoji proxy modulů je dána velmi nenáročnými požadavky na jednotlivé komponenty a dostatečnou úrovní abstrakce v aplikačním programovém rozhraní. Vývojář extenzí není vázán striktními pravidly a může snadno implementovat libovolné protokoly pro jejich použití při testování tlustých klientů, a to bez nutnosti zásahu do jádra nebo jiných existujících modulů. Ty je možné i nadále beze změn využívat ke zpracování nově zaváděných protokolů.

3. 2. 3. 2 Moduly pro interceptory

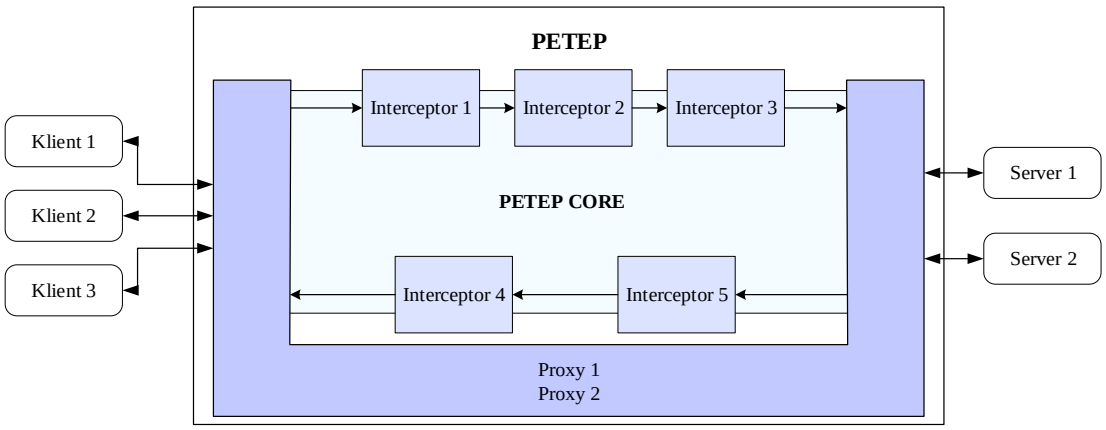
Moduly pro interceptory (z anglického intercept – zachytit) umožňují vytváření procesů pro zpracování PDU zasílaných skrze proxy definované v aplikaci. Lze je tak využít pro logování, modifikaci, zahazování datových balíčků a mnohé další podpůrné činnosti užitečné při penetračních testech. Název interceptor by bylo možné do češtiny volně přeložit jako „zachycovač“ a symbolizuje princip fungování modulu, který je založen na zachycování protokolových datových jednotek probíhajících skrze aplikaci.

Samotný interceptor musí definovat zejména metodu intercept(PDU), která je provolána pro každý datový balíček procházející daným pracovníkem modulu. Prostřednictvím této metody probíhají jednotlivé operace pro zpracování datové jednotky, jež jsou založeny na uživatelské konfiguraci modulu.

Je zcela nezbytné si uvědomit, že skrze aplikaci vedou dva logické toky dat – jeden pro směr mezi klientem a serverem a druhý ve směru opačném, tedy mezi serverem a klientem. Z tohoto důvodu mají oba toky samostatné oddělené fronty interceptorů konfigurovatelné uživatelem aplikace v grafickém rozhraní, přičemž omezení nejsou stanoveny pro minimální ani maximální počet modulů a počty v jednotlivých směrech mohou být rozdílné.

Při příchodu datové jednotky od klienta (prostřednictvím spojení v proxy), předá jádro balíček ke zpracování do fronty příslušného směru. Každý modul pro zpracování dat má

vyhrazeno vlastní vlákno, ve kterém zpracovává PDU ze vstupní fronty a po zpracování je předává dále na frontu výstupní. Po průchodu datové jednotky frontou interceptorů je jednotka zaslána zpět do jádra, kde je předána k odeslání konkrétnímu spojení do sítě. Systém jedné instance proxy složený ze spojení a dalších prvků se stará o přijímání i odesílání téhož balíčku a jádro aplikace slouží pouze pro samotné zpracování dat a propojení jednotlivých modulů – viz následující schéma zobrazující výše popsané principy toku datových jednotek skrze aplikaci.

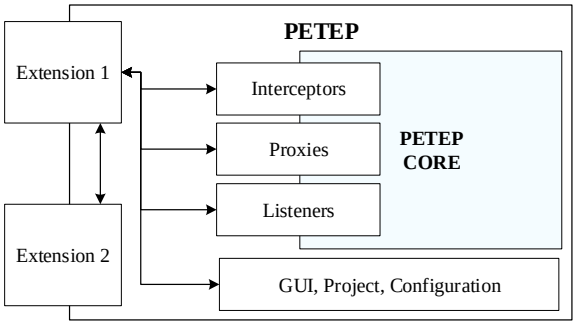


Obrázek č. 17 Schéma modulů pro zpracování dat
(Zdroj: Vlastní zpracování)

3. 2. 4 Rozšíření

Jak vyplývá z analýz rozebraných ve druhé kapitole, jedním ze základních požadavků na aplikaci je její rozšiřitelnost o nové vlastnosti a funkce třetí stranou, a to bez nutnosti zásahu do zdrojového kódu aplikace samotné. K těmto účelům vznikla podpora rozšíření, ke které je využita abstraktní forma tříd pro vývojářské API, díky které programátor extenzí nepotřebuje znát konkrétní definice procesů jádra nebo modulů a je mu poskytnut pouze abstrakcí rámec, jímž jsou vymezeny možnosti pro práci s jádrem aplikace a všemi zásuvnými moduly i ostatními rozšířeními.

Pomocí extenzí může vývojář rozšířit aplikaci o moduly pro zpracování dat, moduly pro proxy nebo například o grafické rozhraní jako jsou záložky v nastavení a hlavním menu aplikace. Rozšiřitelnost však nebrání ani vzniku dalších prvků API, které umožňují přidat novou funkcionalitu i do samotných rozšíření – tento přístup je využit v rámci dvou interních rozšíření Tagger a Modifier, které jsou popsány v samostatné kapitole.



Obrázek č. 18 Schéma rozšiřitelnosti aplikace
(Zdroj: Vlastní zpracování)

Na zjednodušeném schématu rozšiřitelnosti aplikace si lze povšimnout, že rozšíření mohou pracovat jak se samotnou aplikací (grafickým rozhraním, projektem, konfigurací), tak s moduly zasahujícími do jádra a ostatními rozšířeními, a to vše prostřednictvím vývojového API.

Jednotlivá rozšíření se navzájem odlišují unikátním kódem složeným ze znaků, na nějž jsou při řešení persistence vázány jednotlivé konfigurační položky. Mimo kódu dále obsahuje také název, popis a verzi, které slouží pro poskytnutí přehledu uživateli.

Komplexní rozšiřitelnost aplikace si zakládá zejména na následujících aspektech:

- registrace posluchačů událostí (listeners),
- registrace modulů proxy a interceptorů,
- používání pomocných objektů (helpers).

Rozšíření mohou být načítána z interních zdrojů aplikace (interní rozšíření) nebo ze souborového systému uživatele (externí rozšíření) a využívají pro svůj běh API nacházející se ve vyextrahovaném balíčku PetepLib.jar. Ten obsahuje jednotlivé zdrojové kódy vývojářského rozhraní a je distribuován s jednotlivými releasy aplikace.

Balíček je vytvářen pomocí speciálního build skriptu pro Gradle, který do separované knihovny přidává soubory obsahující anotaci PetepAPI (pro zdrojové kódy Javy) a soubory obsahující komentář PetepAPI (pro soubory ve formátu XML). Výsledné extrahované soubory jsou přeneseny do JAR balíčku PetepLib.jar, jenž nachází využití jako knihovna importovatelná do Java projektů a jako modul do aplikace SceneBuilder, která je využívána pro návrh grafického rozhraní. Do této aplikace jsou pomocí importu balíčku přidány interní komponenty aplikace, aby bylo za jejich pomoci možné navrhovat uživatelské prostředí.

Při vytváření externích rozšíření je nutné dodržet požadavek, jenž udává, že v jednom balíčku ve formátu JAR se může vyskytovat pouze jedna jediná třída rozšíření, která musí nést název PetepExtension a musí být součástí Java package s názvem petep. Dodržením těchto zásad je pak možné v aplikaci za pomoci interních funkcí Javy jednoduše provést načtení jednotlivých rozšíření za chodu aplikace.

Interní rozšíření jsou načítány přímo z interních zdrojů aplikace, a tak u nich platí méně striktní pravidla, co se názvů a struktury týče. Jejich nevýhodou však je, že na rozdíl od externích rozšíření jsou neoddělitelnou součástí aplikace a ač nemusí být zavedeny do projektu (nedojde k jejich načtení v prostředí aplikace), budou v každém případě zvyšovat velikost výsledného spustitelného balíčku aplikace. Verze interních rozšíření jsou navíc na pevno svázány s jednotlivými releasy aplikace, a tak není možné používat novější verzi aplikace se starším interním rozšířením a naopak. (Upgrade aplikace s sebou automaticky nese i upgrade interních rozšíření.)

3. 2. 4. 1 Posluchači událostí

Aplikace v průběhu používání přechází mezi stavy, kdy je jádro aktivní a neaktivní. Při aktivním jádře je zakázána modifikace konfigurace modulů, protože jsou spuštěny a prochází skrz ně datový provoz. Při neaktivním jádře lze konfigurovat moduly, jelikož je provoz zastavený. Posluchači událostí zastoupené rozhraním PetepListener poskytují vývojáři možnost provádět činnosti v závislosti na stavu jádra. Pomocí posluchačů událostí jádra může programátor využít následujících metod, jež ohraničují jednotlivé stavy:

- **beforePrepare** – před přípravou jednotlivých instancí modulů ke spuštění,
- **afterPrepare** – po přípravě instancí modulů,
- **beforeStart** – přes samotným spuštěním pracovníků modulů,
- **afterStart** – po úspěšném spuštění pracovníků modulů,
- **beforeStop** – před ukončením pracovníků modulů a činnosti jádra,
- **afterStop** – po ukončení pracovníků modulů a činnosti jádra.

Posluchač událostí vytvořený na základě rozhraní PetepListener poskytuje zároveň pomocný objekt PetepHelper (popsaný v následující kapitole), díky němuž je možné v jednotlivých metodách provádět operace přímo nad běžícím jádrem aplikace, a tak

posluchače událostí nalézají využití zejména v rozšířeních, které při startu jádra poskytují uživateli podpůrné funkce pro práci se spojeními či zasílanými daty.

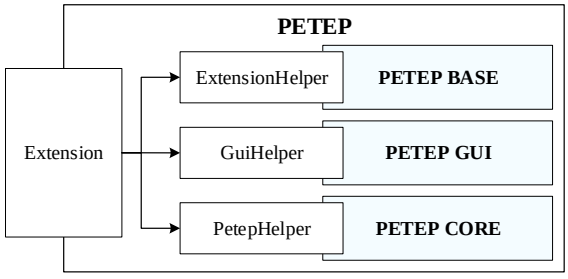
Mimo posluchače událostí jádra existují v aplikaci také dva velmi specifické posluchače (ExtensionInitListener a ExtensionGuiInitListener), které poskytují rozšířením možnost provádět činnosti před a po inicializaci ostatních rozšíření a inicializaci jejich grafických rozhraní. Jejich využití spočívá v možnosti zahájit komunikaci mezi jednotlivými rozšířeními před či po samotném načtení všech částí rozšíření.

3. 2. 4. 2 Pomocné objekty

Pomocné objekty (helpers) jsou abstrakcí mezi rozšířením a interním fungováním aplikace. Umožňují vývojářům provolávat metody uvnitř jádra aplikace bez potřeby bližší znalosti jejich konkrétní implementace. Mezi rozšířením a komponentami aplikace tak vzniká potřebná míra abstrakce, která je v případě vývoje rozšiřitelných systémů prakticky nutná. Nejen, že díky ní nedochází k zatížení vývojáře nepodstatnými informacemi, ale zároveň je významně snazší provádět úpravy v jádře, aniž by došlo k narušení kompatibility jádra s rozšířeními, jelikož se tyto úpravy ve velké většině případů dějí pouze pod povrchem abstraktních prvků.

Pro účely abstrakce práce s komponentami aplikace existují tři druhy pomocných objektů rozdělené dle jejich účelu:

- **ExtensionHelper** – registrace modulů, vytváření listenerů a práce s ostatními zavedenými rozšířeními,
- **GuiHelper** – registrace grafických prvků,
- **PetepHelper** – práce s běžícím jádrem aplikace.



Obrázek č. 19 Schéma pomocných objektů
(Zdroj: Vlastní zpracování)

K pomocným objektům se programátor dostane na místech, která jsou k nim vázaná na základě interní logiky zavádění modulů a rozšíření. Například uvnitř pracovníků

jednotlivých modulů je vývojáři přidělen pomocný objekt pro jádro aplikace (PetepHelper), který mohou využít pro pokročilou práci s běžícími instancemi a jádrem, a při inicializaci rozšíření je vývojáři přidělen pomocný objekt pro registraci modulů (ExtensionHelper), pomocí něhož může zavést jednotlivé rozšiřující prvky do aplikace nebo komunikovat s ostatními rozšířeními.

3. 2. 5 Konfigurace modulů a rozšíření

Konfigurace modulů a rozšíření je řešena dobrovolnou implementací rozhraní Configurable, které umožňuje modulům a rozšířením ukládat a načítat nastavení přímo z konfiguračních souborů aplikace, jenž jim dle jejich typu přísluší.

Práce s konfigurací vývojáře žádným způsobem nezatěžuje procesem použitým pro ukládání, načítání a formátování konfiguračního souboru, jelikož je celé zpracování založené na autonomním serializování a deserializování Java objektů pomocí knihovny Gson. Programátor prostřednictvím rozhraní Configurable předává aplikaci pro uložení svůj interní konfigurační objekt, aplikace pro uložení provede serializaci pomocí knihovny Gson a výsledný řetězec ve formátu JSON uloží do příslušného konfiguračního souboru. Deserializace následně funguje obdobným způsobem a programátorovi je v kódu předán opět jeho vlastní konfigurační objekt.

```
6  @PetepAPI
7  public interface Configurable<C> {
8    /** Returns configuration to be saved. */
9    C saveConfig();
10
11   /** Loads configuration. */
12   void loadConfig(C config);
13 }
```

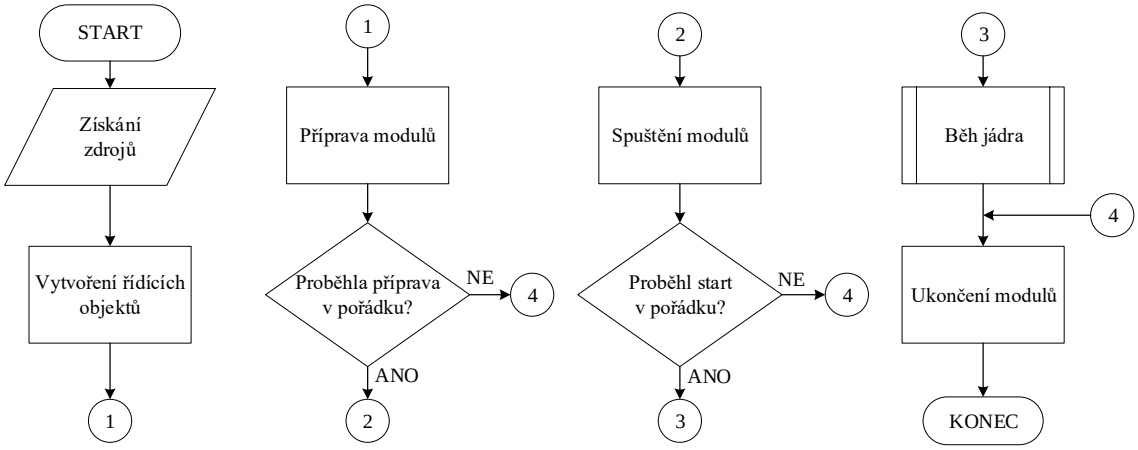
Obrázek č. 20 Rozhraní Configurable
(Zdroj: Vlastní zpracování)

Mimo rozhraní Configurable mají moduly i rozšíření přístupné také rozhraní Storable, které na rozdíl od rozhraní Configurable slouží pro ukládání libovolných dat, tedy nejen konfiguračních, ale například také operačních. Princip persistence datových objektů je v případě obou rozhraní navržen stejným způsobem a lze je obě používat současně.

3. 2. 6 Spouštění jádra

Běh aplikace s grafickým prostředím má ve výchozím stavu jádro automaticky deaktivované a aplikace je v konfiguračním režimu, v němž může uživatel vytvářet a nastavovat jednotlivé moduly. Po nastavení modulů uživatel pomocí tlačítka START

v nastavení spustí jádro aplikace a je aktivován proces znázorněný pomocí následujícího vývojového diagramu.



Obrázek č. 21 Vývojový diagram procesu jádra
(Zdroj: Vlastní zpracování)

Na výše uvedeném zjednodušeném procesu jádra je možné zpozorovat, že samotný start má dvě části – přípravnou a spouštěcí, po nichž jádro běží až do chvíle, kdy uživatel stiskne tlačítko pro ukončení jádra, nebo dojde k jiné události, která zapříčiní jeho ukončení (například vypnutí aplikace či vznik kritické výjimky). Pokud během startu selže alespoň jedna z částí, dojde k okamžitému ukončení všech modulů a jádro zůstane deaktivované.

Obě části startu (příprava a spouštění) provádí paralelní volání odpovídajících metod podřízených instancí modulů – pracovníků (prepare a start), a zároveň dochází k hlášení jednotlivých událostí posluchačům zmíněným v předcházejících kapitolách.

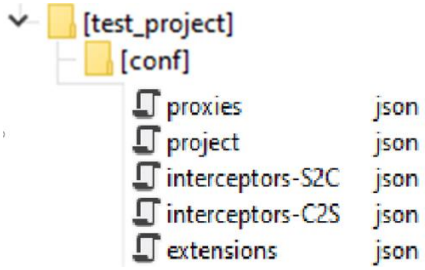
3. 3 Projekty

Projekt zastřešuje konkrétní konfiguraci aplikace určenou pro jeden konkrétní účel a obsahuje kompletní uživatelské nastavení a soubory používané při jednom konkrétním testu. Pro správu projektů je součástí aplikace takzvaný Wizard, v němž lze vytvářet, načítat a odstraňovat jednotlivé projekty a konfigurovat jejich nastavení, a to zejména seznam použitých rozšíření, která mají být načtena při spuštění aplikace.

Projekt samotný je vymezen umístěním jeho adresáře v souborovém systému uživatele a v případě přímého spouštění projektu v kontextu aplikace je tento adresář použit jako hlavní vstupní parametr.

3. 3. 1 Základní struktura projektu

Základní struktura projektu je pevně stanovena, ale je na uživateli, zda ji rozšíří o další soubory a adresáře. Její aktuálně jedinou povinnou položkou je adresář konfigurace obsahující pět souborů s nastavením aplikace.



Obrázek č. 22 Adresářová struktura projektu
(Zdroj: Vlastní zpracování)

Šablona pro projekt se nachází ve složce aplikace a je možné ji modifikovat, čehož může uživatel využít, pokud chce rozšířit pevně danou strukturu projektu o další výchozí soubory a adresáře a usnadnit si vytváření budoucích projektů. Velmi praktické je například dopředné vytvoření adresářů pro logovací soubory, poznámky a výstupy z penetračních testů, které bude uživatel pomocí aplikace provádět.

3. 3. 2 Konfigurace projektu

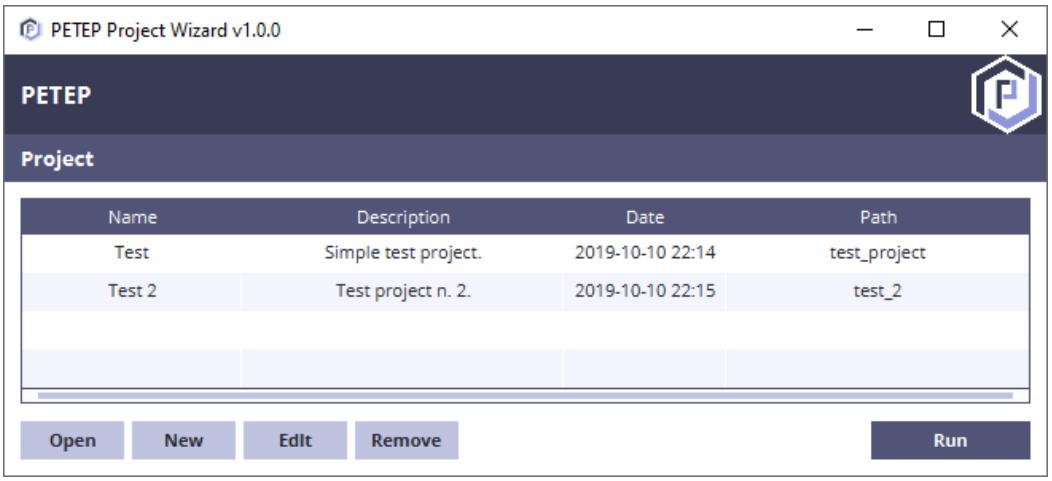
Konfigurační soubory jsou v rámci celé aplikace uskutečněny pomocí formátu JSON, který je čitelný a srozumitelný i pro člověka, a přitom je podstatně kompaktnější než formát XML. Samotná konfigurace projektu v adresáři „conf“ obsahuje vždy minimálně následujících pět souborů s nastavením potřebným pro chod aplikace:

- **project.json** – základní nastavení projektu,
- **extensions.json** – seznam používaných interních a externích rozšíření,
- **proxies.json** – konfigurace proxy,
- **interceptors-C2S.json** a **interceptors-S2C.json** – konfigurace interceptorů.

Pro práci s JSON soubory je využita knihovna GSON od společnosti Google, jež umožňuje serializaci a deserializaci Java objektů z a do textových struktur a značně usnadňuje práci s tímto populárním formátem. Logika zpracování konfigurací je pak z důvodu dlouhodobé udržitelnosti oddělena od zbytku aplikace, aby bylo kdykoli v budoucnu možné snadno přejít na jinou knihovnu nebo jiný datový formát pro zajištění persistence konfiguračních a operačních dat aplikace.

3. 3. 3 Správa projektů

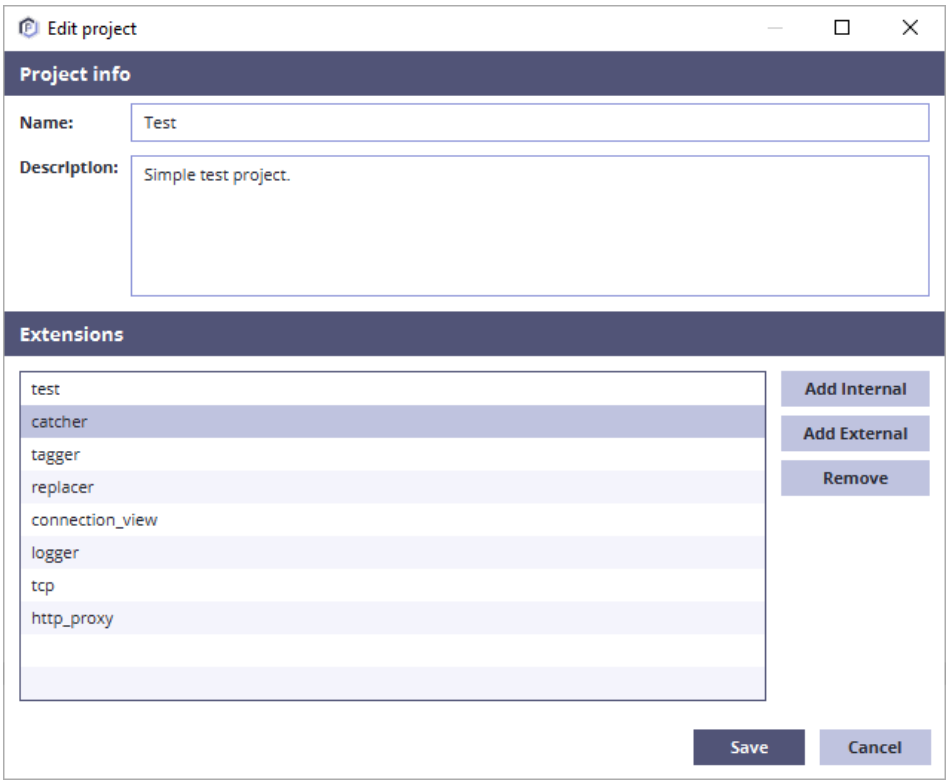
Správa projektů je uživateli usnadněna prostřednictvím takzvaného projektového Wizarda, který provádí uživatele spravováním projektů – jejich vytvářením, editací a odstraňováním – a jednotlivé projekty je z něj možné spustit.



Obrázek č. 23 Konfigurační wizard projektu
(Zdroj: Vlastní zpracování)

Projektový Wizard ukládá seznam projektů do souboru „petep.json“ v místě spuštění aplikace ve formě JSON pole obsahujícího řetězce s cestami k adresářům jednotlivých projektů.

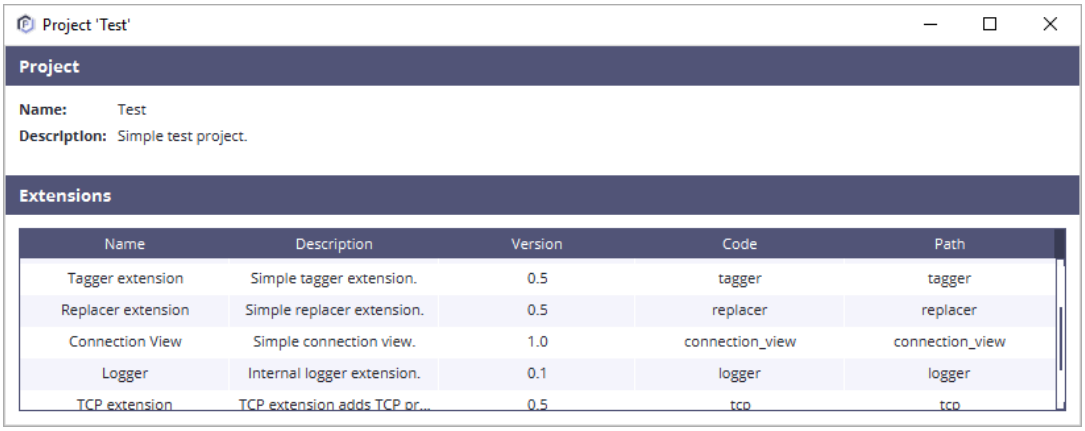
Konfigurace projektu v prostředí Wizardu obsahuje nastavení používaných rozšíření, která mají být při spuštění aplikace načtena do paměti procesu a definici názvu a popisu projektu, pomocí kterých se může uživatel snadno orientovat v seznamu s více projekty. Tyto konfigurační položky jsou následně ukládány v konfiguračním adresáři konkrétního projektu.



Obrázek č. 24 Editace projektu
(Zdroj: Vlastní zpracování)

Po spuštění aplikace s konkrétním projektem již není možné spravovat seznam rozšíření, který lze vidět na předešlém snímku obrazovky, jelikož jsou rozšíření načítána při samotném startu. Pokud se uživatel rozhodne pro změnu použitých rozšíření, musí nejdříve ukončit projekt a přejít zpět do konfiguračního Wizarda, kde může daný projekt modifikovat, a poté opět spustit s novým nastavením.

Seznam aktuálně aktivních rozšíření, dostupný přes prvek menu (Menu – Project – Info), zobrazí přehled rozšíření včetně podrobnějších informací o nich.



Obrázek č. 25 Informace o projektu
(Zdroj: Vlastní zpracování)

Informace zobrazené o rozšířeních v projektovém dialogu vyplňují sami vývojáři při tvorbě svých rozšíření, a to konkrétně při definici jednotlivých metod deklarovaných ve třídě Extension.

3. 4 Grafické rozhraní

Aplikace je ve výchozím režimu stavěna pro používání grafického rozhraní postaveného na knihovně JavaFX a její grafické prostředí je navrženo v souladu s rozšiřitelností a modularitou. Výchozí vzhled rozhraní je upraven pomocí kaskádových stylů, aby držel jednotnou formu a odlišil se od ostatních aplikací stavících na použité knihovně. Právě pro tyto účely jsem zvolil specifickou skupinu barev v odstínech pohybujících se na pomezí modré a fialové barvy, které jsou v rámci celé aplikace jednotně používány.

Pro grafické rozhraní aplikace a webovou prezentaci jsem vytvořil následující vektorové logo, které je používané především jako ikona aplikace a dialogů, a tak musí být dostatečně jednoduché pro dobrou rozpoznatelnost i v menším rozlišení.

Obrázek č. 26 Logo aplikace PETEP
(Zdroj: Vlastní zpracování)

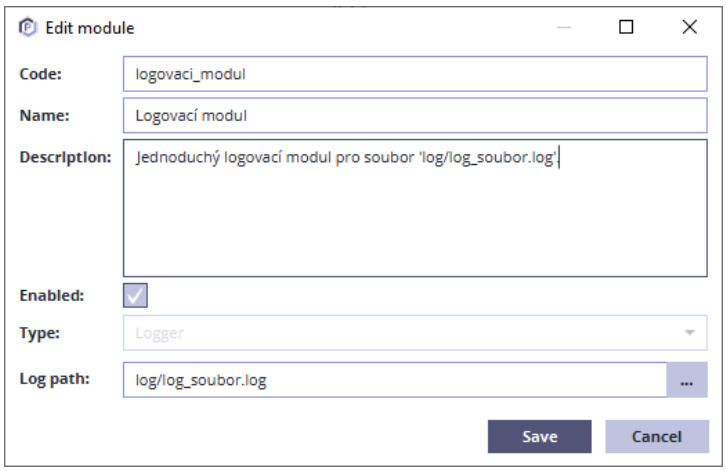
3. 4. 1 Grafické komponenty

Rozšiřitelnost a modularita přináší do vývoje a návrhu grafického prostředí požadavky na možnost vytvářet a rozšiřovat grafické komponenty přes API dostupné všem rozšířením. Z těchto důvodů se stalo nutností vytvoření zejména následujících čtyř grafických komponent:

- **ConfigPane** – konfigurační panel,
- **PduMetadataPane** – panel pro metadata protokolové datové jednotky,
- **BytesEditor** – jednoduchý editor bajtů dat,
- **PduEditor** – editor pro protokolovou datovou jednotku.

3. 4. 1. 1 Konfigurační panel

Konfigurační panel (ConfigPane) je abstraktní třída, jejímž rozšířením moduly vytváří vlastní grafický prvek pro nastavení. Její existence dává aplikaci možnost zobrazovat uživateli komponentu pro konfiguraci závislou na použitém rozšíření i modulu, a to bez znalosti struktury či datového obsahu této konfigurace.



Obrázek č. 27 Konfigurace modulu
(Zdroj: Vlastní zpracování)

Na výše uvedeném obrázku lze vidět konfigurační panel, v němž je uživateli zobrazen dialog bez rozdílu mezi nastavením daným aplikací a nastavením specifickým pro modul. Celá část konfigurace od identifikačního kódu po typ modulu je stanovena a vyžadována aplikací a cesta pro logovací soubor je do GUI vygenerována prostřednictvím interního rozšíření pro logování (Logger).

Vývojář může zobrazení konfiguračního panelu vyvolat implementací rozhraní Configurator do továrny modulů, a to jak pro modul proxy, tak pro modul interceptoru.

Mimo moduly proxy a moduly interceptorů je konfigurační panel používán i některými interními rozšířeními, která umožňují vývojářům vytvářet vlastní pravidla používaná v těchto rozšířeních, jako je Tagger a Modifier.

3. 4. 1. 2 Editor PDU

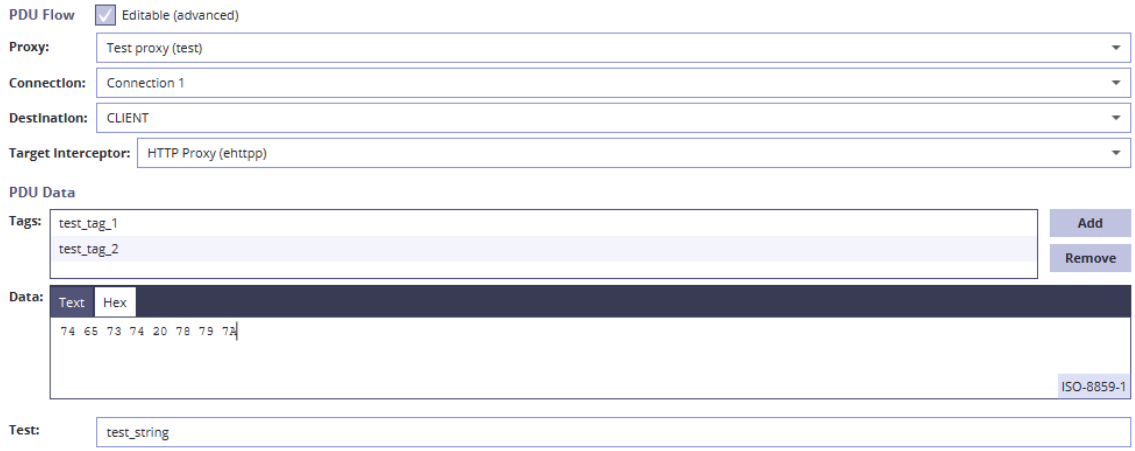
Pro editaci jednotek PDU vznikla komponenta PduEditor, která obsahuje tři vnořené komponenty pro práci s jednotlivými částmi balíčku:

- editor toku,
- editor dat,
- editor metadat.

Editor toku slouží ke správě informací o proxy, spojení, směru a následujícím interceptoru, v němž má zpracování PDU pokračovat. Ve výchozím nastavení je celá tato část deaktivována, jelikož vyžaduje větší zkušeností s používáním aplikace i její architekturou a ve většině případů není její použití nutné.

Editor dat je zároveň samostatnou komponentou (PduEditor) a umožňuje upravovat datovou část PDU balíčků pomocí textového a hexadecimálního editoru.

Editor metadat se liší v závislosti na nadřazeném protokolu konkrétního PDU a je definován továrnou proxy modulů. Metadata mohou obsahovat data specifická pro konkrétní aplikační protokol jako jsou například HTTP hlavičky a verze.



Obrázek č. 28 Editor PDU
(Zdroj: Vlastní zpracování)

Na výše uvedeném obrázku si lze povšimnout, že prvek pocházející z testovacího proxy modulu označený štítkem „Test:“ pod editorem dat, je součástí celého editoru. Rozšiřitelnost aplikace o nové protokoly tak není omezena na pouhé přidávání neviditelné logiky zpracování protokolů, ale dovoluje zároveň doplnění vlastních grafických prvků pro snazší práci s těmito protokoly.

Editor PDU je zahrnut v API aplikace a lze jej používat ve všech rozšířeních. Jeho velkou výhodou je, že není nutné znát konkrétní protokoly, které bude uživatel v editoru spravovat, a je možné použít tento jednotný prvek pro všechny dohromady. Panel metadat se totiž do editoru načítá automaticky při přiřazení jednotky PDU pro editaci.

3. 4. 2 Spuštění aplikace bez GUI

Aplikace je vytvořena pro využívání s grafickým rozhraním, avšak umožňuje také spuštění bez něj, které provede automatické spuštění všech modulů dle uživatelského nastavení definovaného prostřednictvím GUI či manuální úpravou JSON souborů.

Pro spuštění aplikace bez grafického prostředí je nutné zadat jako spouštěcí parametry aplikace cestu k projektu a parametr „--nogui". Při zadání těchto parametrů aplikace spustí zvolený projekt a aktivuje jádro s využitím existujícího nastavení.

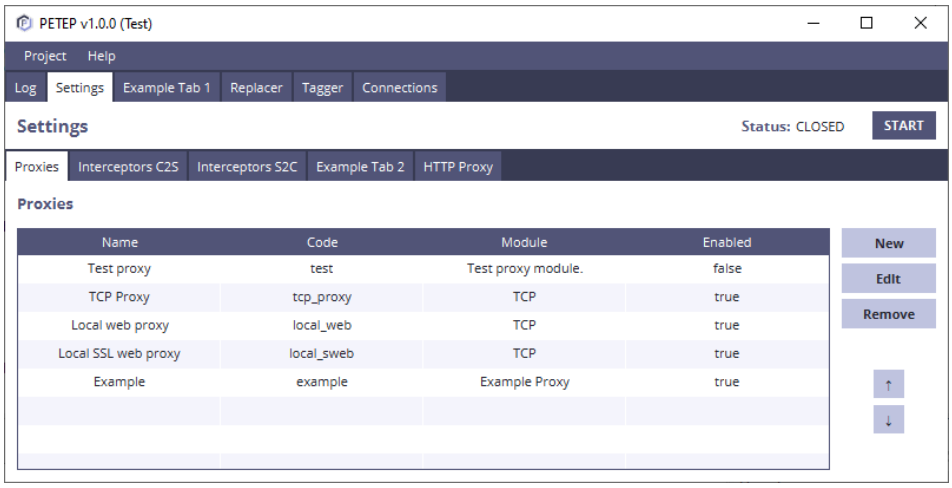
Při spuštění aplikace bez grafického uživatelského rozhraní neproběhne v rozšířeních načítání ani inicializace mnohých zdrojů, čímž se aplikace stává podstatně úspornější, a toho lze využít nejen v případě, kdy uživatel nemá možnost grafické rozhraní zobrazit, ale i v případě, kdy uživatel plánuje použít pouze negrafické funkce aplikace a potřebuje šetřit výpočetními zdroji.

3. 4. 3 Grafické rozhraní pro konfiguraci

Jak již bylo zmíněno v kapitole o konfiguraci projektu, nastavení se skládá z 5 souborů ve formátu JSON. Dva z těchto souborů jsou konfigurovány před samotným otevřením projektu pomocí projektového Wizarda a následující tři mohou být konfigurovány za běhu aplikace:

- proxies.json,
- interceptors-C2S.json,
- interceptors-S2C.json.

Konfigurace proxy je možná prostřednictvím záložky „Settings" a podzáložky „Proxies", kde se zobrazuje seznam aktuálně nastavených proxy včetně základních informací o nich. Uživatel zde může vytvářet, upravovat a odebírat jednotlivé moduly proxy, popřípadě měnit jejich pořadí, což však nijak neovlivní chod jádra a slouží pouze pro přehlednost.



Obrázek 2 Nastavení proxy
(Zdroj: Vlastní zpracování)

Konfigurace interceptorů je možná prostřednictvím záložky „Settings“ a podzáložek „Interceptors C2S“ a „Interceptors S2C“, které obsahují seznam nakonfigurovaných modulů interceptorů pro jednotlivé směry. Konfigurační možnosti jsou totožné jako u proxy modulů na výše uvedeném obrázku, avšak v případě interceptorů změna pořadí modulů zásadně ovlivní fungování aplikace, jelikož jednotlivé moduly zpracovávají protokolové datové jednotky vždy v nastaveném pořadí.

Mimo výše zmíněných konfiguračních záložek, jež jsou součástí aplikace, lze přidávat vlastní záložky pro jednotlivá rozšíření. Na výše uvedeném snímku nastavení jsou takové záložky dvě – jedna ukázková („Example Tab 2“) a druhá pro konfiguraci externí HTTP proxy („HTTP Proxy“), která je blíže vysvětlena v kapitole o interních rozšířeních.

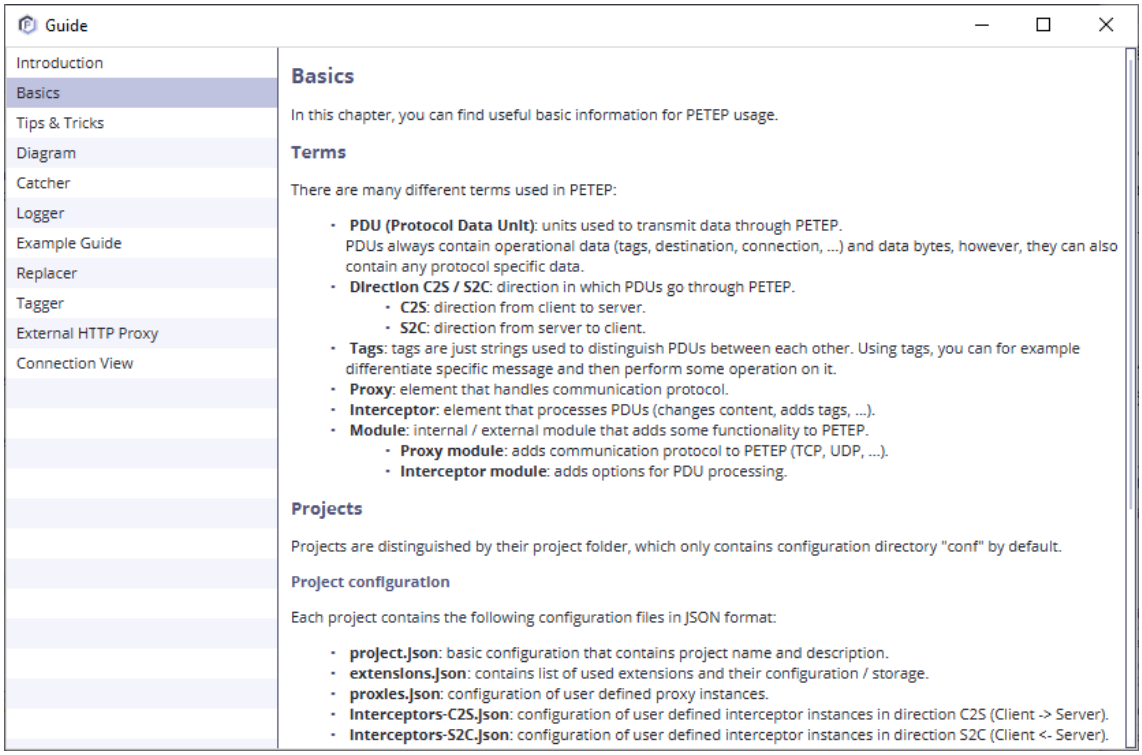
3. 5 Příručka

Součástí aplikace je příručka zobrazitelná pomocí klávesy F1, popřípadě přes tlačítko v hlavním menu aplikace (Help – Guide). Ve výchozím stavu příručka obsahuje následující stránky:

- úvod,
- základy,
- tipy a triky,
- diagram.

Princip příručky staví na dialogovém okně rozděleném na seznam příruček a komponentě pro zobrazování webových stránek. Jednotlivé příručky jsou poté načítány jako webové

stránky z textového řetězce, který může být získán ze statického souboru HTML, nebo může být dynamicky sestaven v příslušné metodě v Java kódu.



Obrázek č. 29 Příručka v aplikaci
(Zdroj: Vlastní zpracování)

3. 5. 1 Úvodní stránka

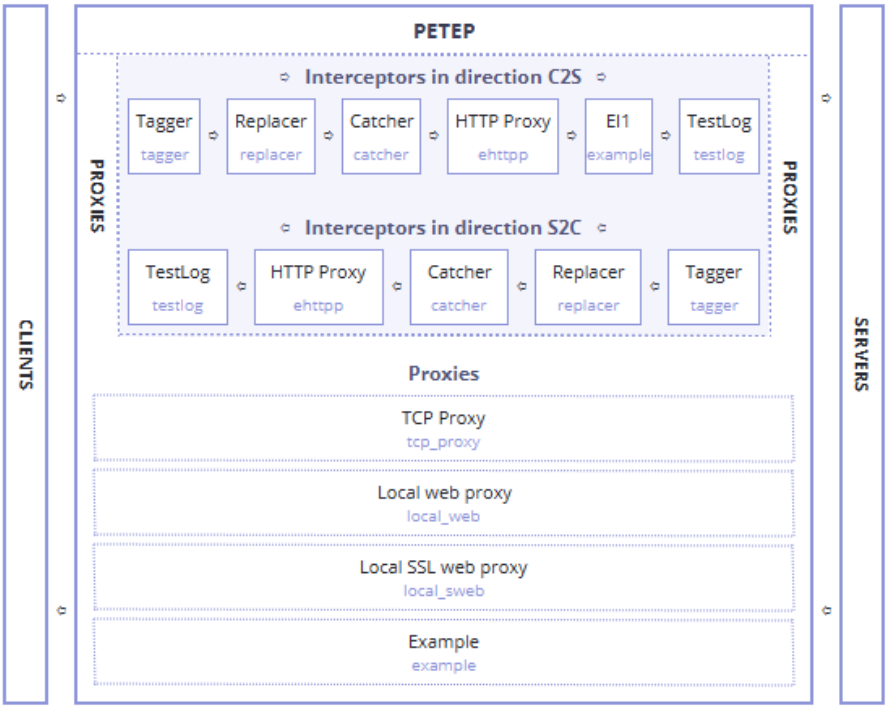
Úvodní stránka příručky obsahuje základní informace o aplikaci jako je popis aplikace, informace o autorovi a seznam interních rozšíření, která může uživatel ve spuštěné verzi aplikace využívat.

3. 5. 2 Stránka se základy

Stránka se základy je nejdůležitější částí nápovědy pro nové uživatele aplikace, jelikož vysvětluje nejzákladnější a nejzásadnější funkce nástroje a používané pojmy a zkratky, jejichž znalost je nutná pro správnou orientaci v prostředí aplikace.

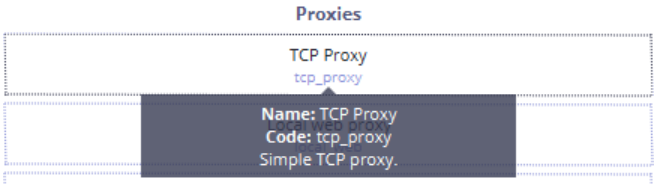
3. 5. 3 Stránka diagramu

Stránka diagramu vykresluje aktuální nastavení modulů aplikace ve formě přehledného interaktivního obrázku, v němž lze vyčíst informace o jednotlivých modulech aplikovaných v jádře.



Obrázek č. 30 Diagram modulů v aplikaci
(Zdroj: Vlastní zpracování)

Názvy a kódy zobrazené v diagramu jsou zadávány uživatelem při nastavování jednotlivých modulů, další informace může uživatel zadat do popisu modulu, který je zobrazen při najetí kurzorem nad daný modul (viz následující obrázek).



Obrázek č. 31 Zobrazený popis modulu
(Zdroj: Vlastní zpracování)

Celý diagram je dynamicky sestavován pomocí Java kódu na základně aktuálního nastavení aplikace, a poskytuje tak užitečný přehled při používání aplikace během testů tlustých klientů.

3. 5. 4 Příručky rozšíření

Příručky lze rozšířit o další stránky jejich registrací přes pomocné objekty v rozšířeních, což umožňuje jednotlivým rozšířením přidávat uživatelské návody pro svoji vlastní funkcionalitu. Všechna interní rozšíření, která jsou součástí této práce, přidávají vlastní stránku obsahující základní informace potřebné pro jejich správné a bezproblémové používání.

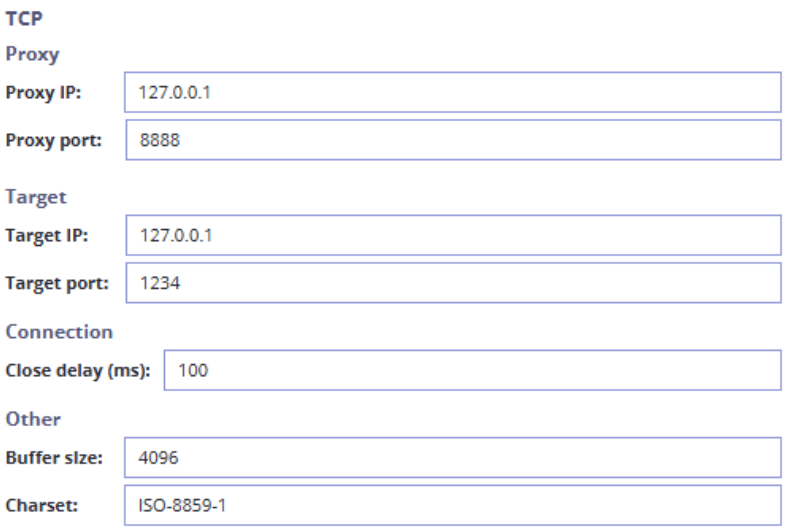
3. 6 Interní rozšíření

Interní rozšíření do aplikace zavádí konkrétní funkcionalitu, jelikož čisté jádro je pouze základem, který data předává ze strany na stranu a zajišťuje propojení jednotlivých modulů. Interní rozšíření fungují na stejném principu jako externí rozšíření a jediný rozdíl je v jejich pevném umístění uvnitř kódu aplikace, což usnadňuje jejich distribuci s aplikací, ale zároveň nepřekáží možnosti deaktivovat jejich načítání v jednotlivých projektech na základě uživatelských požadavků.

3. 6. 1 Rozšíření TCP

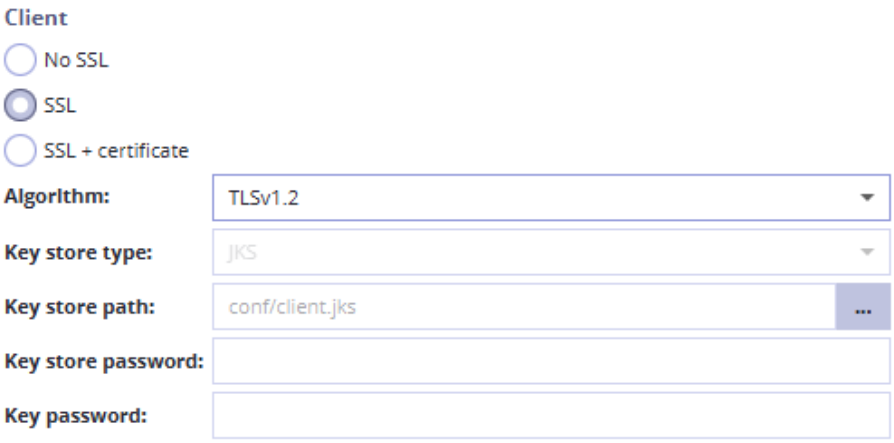
Protokol TCP je jeden z nejrozšířenějších protokolů, se kterými se během testů tlustých klientů setkáváme, a bývá velmi často podkladem pro aplikační protokoly používané v komerční i nekomerční sféře.

Do aplikace je protokol TCP zaveden pomocí interního rozšíření a při vytváření TCP proxy provádí uživatel nastavení cílových a zdrojových IP adres a portů, definuje velikost datového bufferu pro protokolové datové jednotky, volí výchozí znakovou sadu a nastavuje zpoždění, se kterým mají být ukončena otevřená spojení, když dojde k uzavření jedné ze stran spojení. (Tento parametr zabraňuje nedokončení datových přenosů probíhajících těsně před ukončením spojení, a je tak velmi významný například v protokolu HTTP, kde běžně dochází k okamžitému ukončení komunikace ihned po přenesení požadovaných dat.)



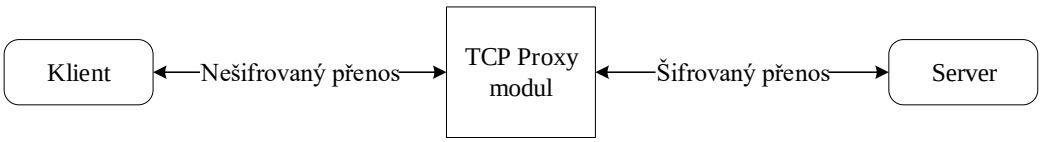
Obrázek č. 32 Nastavení proxy modulu TCP
(Zdroj: Vlastní zpracování)

Mimo podporu pro nešifrovanou TCP komunikaci přidává rozšíření zároveň podporu pro SSL/TLS šifrování, a to s podporou jak pro serverový, tak pro klientský certifikát.



Obrázek č. 33 Nastavení SSL/TLS pro klienta proxy
(Zdroj: Vlastní zpracování)

Šifrování je navíc možné aplikovat i samostatně na jedné straně proxy, což lze v praxi uplatnit v případech, kdy mnozí tlustí klienti mají možnost vypnout šifrování, ale servery mají nešifrované spojení zakázané. Pro takový případ lze deaktivovat šifrování na straně ke klientovi a aktivovat šifrováni na straně k serveru, jako je znázorněno na následujícím diagramu.



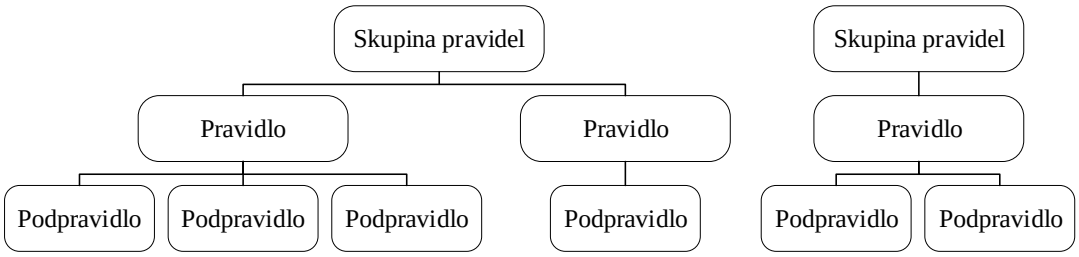
Obrázek č. 34 Kombinace nešifrovaného a šifrovaného přenosu
(Zdroj: Vlastní zpracování)

3. 6. 2 Rozšíření Tagger

Protokolové datové jednotky obsahují množinu takzvaných tagů (štítky, jmenovky), které slouží k označování PDU pro jejich rozlišení ze strany aplikace i ze strany uživatele. Přidělování tagů může zajišťovat libovolný modul, avšak nejdůležitější je, aby tuto možnost měl sám uživatel aplikace, a proto vzniklo rozšíření Tagger, které umožňuje spravovat pravidla pro označování jednotlivých datových jednotek probíhajících skrze tento modul.

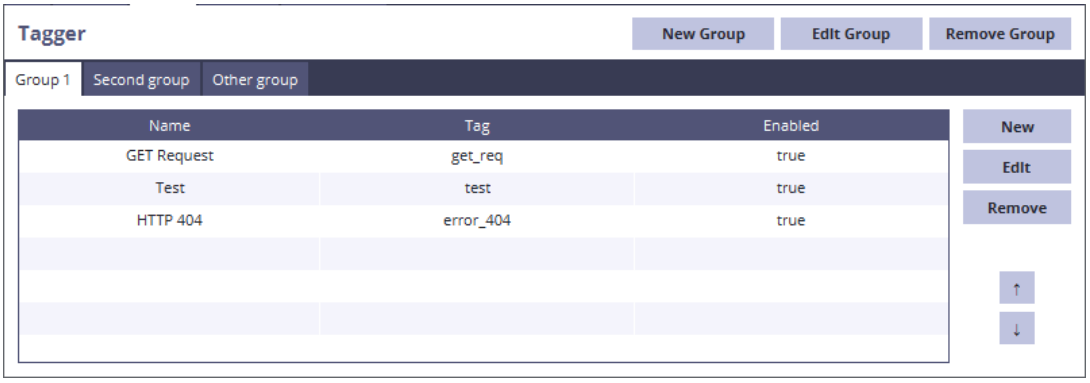
Tagger zavádí speciální skupiny pravidel definované uživatelem, do nichž lze přidávat jednotlivá pravidla sestávající z podmínek opírajících se o vyhodnocovaná podpravidla,

která udávají konkrétní funkcionalitu testování PDU, jako je měření počtu přenášených bajtů či vyhledávání sekvencí znaků v testované jednotce.



Obrázek č. 35 Schéma systému pravidel pro označování PDU
(Zdroj: Vlastní zpracování)

Uvnitř správy skupin pravidel může uživatel vytvářet, editovat a odstraňovat pravidla, popřípadě měnit pořadí, v němž jsou jednotlivá pravidla aplikována.



Obrázek č. 36 Správa skupiny pravidel pro označování PDU
(Zdroj: Vlastní zpracování)

Nastavení konkrétního pravidla spočívá ve stanovení názvu, popisu, přiřazovaného štítku (tagu), definování seznamu podpravidel a sestavení logického výrazu.

Logický výraz definuje závislosti mezi jednotlivými podpravidly a je vyhodnocen jako podmínka, při jejímž splnění dojde k přiřazení štítku testované protokolové datové jednotce. Výraz se může skládat z následujících prvků:

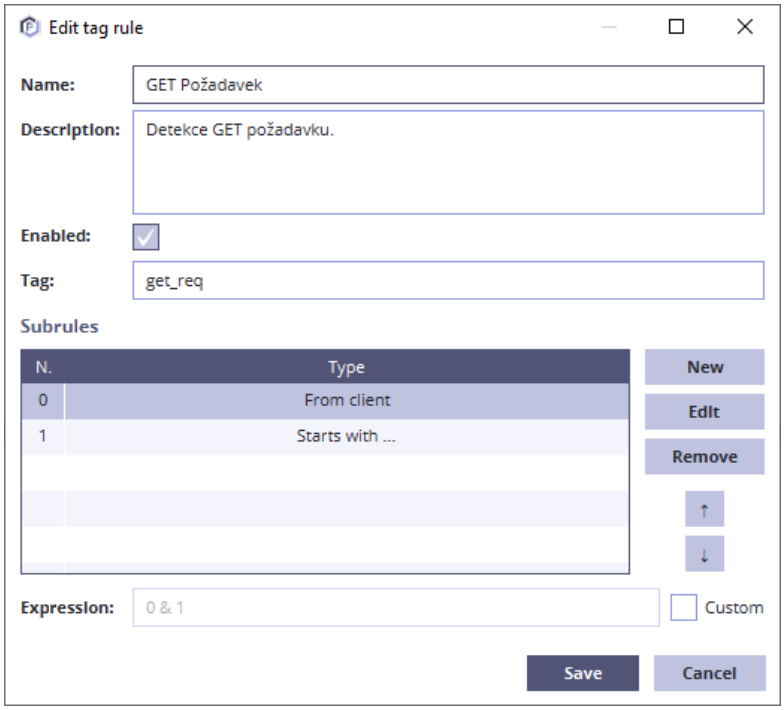
- čísla podpravidel (znaky „0123456789“),
- logické AND (znak „&“),
- logické OR (znak „|“),
- logické XOR (znak „^“),
- logické NOT (znak „!“),
- závorky pro prioritizaci (znaky „{[()]}”).

Ve výchozím režimu se podmínka generuje automaticky jako logický součin, a pro přidělení štítku tak musí dojít ke splnění všech podpravidel. Po úpravě může vzniknout složitější pravidlo, jako lze vidět na následujícím snímku obrazovky.

Obrázek č. 37 Ukázkový výraz pro označovací pravidla
(Zdroj: Vlastní zpracování)

Logický výraz je na pozadí převeden do objektové struktury založené na návrhovém vzoru zvaném „Interpreter pattern“, jenž umožňuje velmi rychlé vyhodnocení podmínek, aniž by musel být opakovaně zpracováván vstupní řetězec. Kvůli další optimalizaci jsou vstupní testy reprezentované soustavou podpravidel uloženy prostřednictvím třídy PredicateExpression, která neprovede test, dokud není nutné zjistit výslednou hodnotu, a při prvním provedení si uchová výsledek pro případ opakovaného výskytu tohoto testu v podmínce. Díky těmto optimalizačním postupům je možné používat systém pravidel i pro komunikaci v reálném čase, aniž by do ní bylo zavedeno omezující zpoždění.

Při úspěšném vyhodnocení podmínky je PDU přidělen tag, který může mimo vlastní uživatelské štítky být také formou speciálního štítku (drop), který způsobí okamžité zahození protokolové datové jednotky.

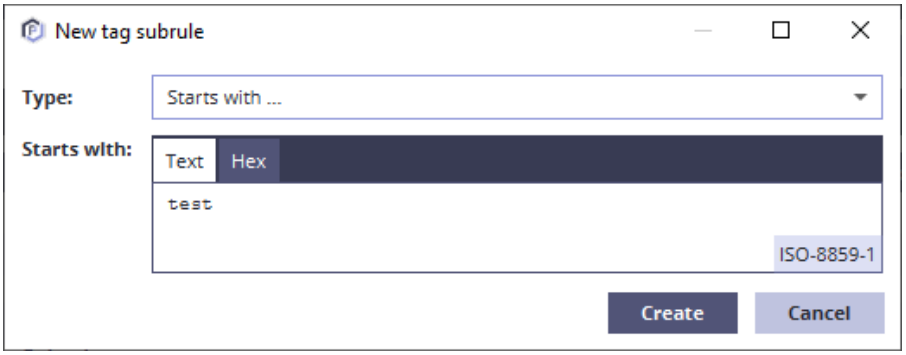


Obrázek č. 38 Správa pravidla pro označování PDU
(Zdroj: Vlastní zpracování)

Podpravidla slouží k definování konkrétního vyhodnocovacího testu prováděného nad PDU (detekce obsahu, původu a mnoho dalších) a jejich počet není nikterak omezen.

V současnosti existují v základu následující typy podpravidel:

- **contains** – test na přítomnost hodnoty v datové části PDU,
- **starts_with/ends_with** – test, zda datová část PDU začíná/končí na určitou hodnotu,
- **size** – test velikosti datové části PDU,
- **has_tag** – test přítomnosti štítku,
- **destination** – pravidlo pro směr, jímž musí PDU směřovat,
- **proxy** – pravidlo pro proxy, ze které musí PDU pocházet.



Obrázek č. 39 Editace „podpravidla“ pro označování PDU
(Zdroj: Vlastní zpracování)

Typy podpravidel pro označování protokolových datových jednotek lze rozšiřovat prostřednictvím interních i externích rozšíření, která pomocí metody beforeInit provedou registraci svých vlastních typů podpravidel.

3. 6. 3 Rozšíření Modifier

Rozšíření Modifier zavádí do procesu zpracování dat možnost vytvářet skupiny pravidel pro automatické modifikace přenášených protokolových datových jednotek, a to například nahrazováním definované části datového pole PDU definovaným polem bajtů.

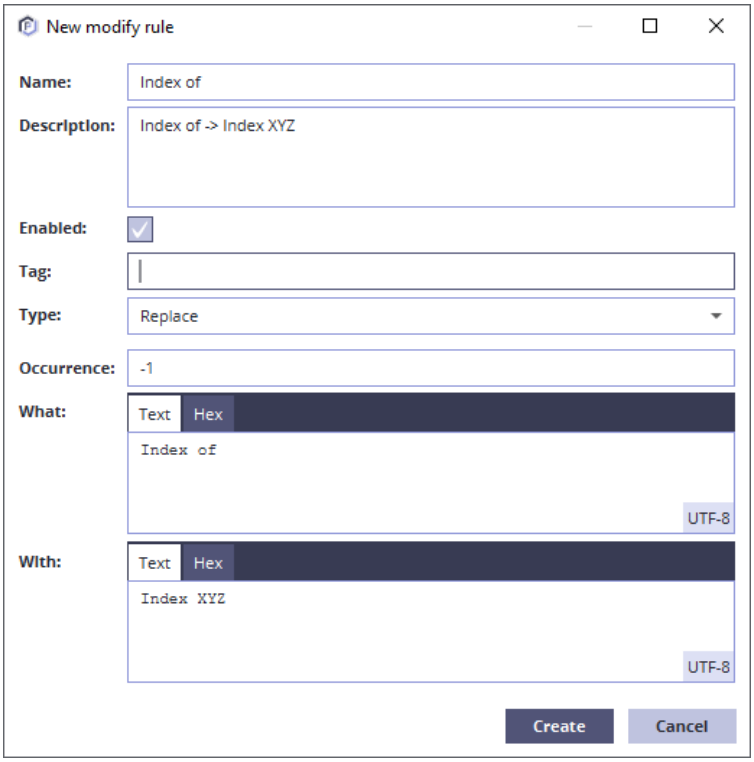
Princip fungování skupin pravidel je totožný jako u rozšíření Tagger a liší se až samotná pravidla pro modifikace, která obsahují mimo základních popisných informací také hodnotu štítku, jež definuje, v jakém PDU má být daná modifikace provedena.

Samotná logika modifikace protokolových datových jednotek je oddělena do samostatných celků, které mohou být do modulu zaváděny jak interními rozšířeními, tak

formou externích rozšíření, jež mohou přidávat vlastní typy modifikačních pravidel. Jde tak o určitou formu rozšiřitelnosti rozšíření, která poskytuje silný podpůrný nástroj pro zavádění nových protokolů vývojářům třetích stran, jelikož významně zjednodušuje a urychluje proces implementace funkcionality pro uživatelské operování s metadaty PDU, která se pro jednotlivé protokoly liší.

Součástí rozšíření Modifier je základní modifikační pravidlo zvané Replace, které dává uživatelům možnost zavádět do systému jednoduchá pravidla automaticky nahrazující stanovenou posloupnost bajtů jinou posloupností. Pro nastavení pravidla je pak nutné, aby uživatel nakonfiguroval následující parametry:

- **occurrence** – kolikátý výskyt má být nahrazen, přičemž pořadí začíná od 0 a pro všechny výskyty je potřeba zadat hodnotu -1,
- **what** – sekvence bajtů, která má být nahrazena,
- **with** – bajty, kterými má být sekvence what nahrazena.



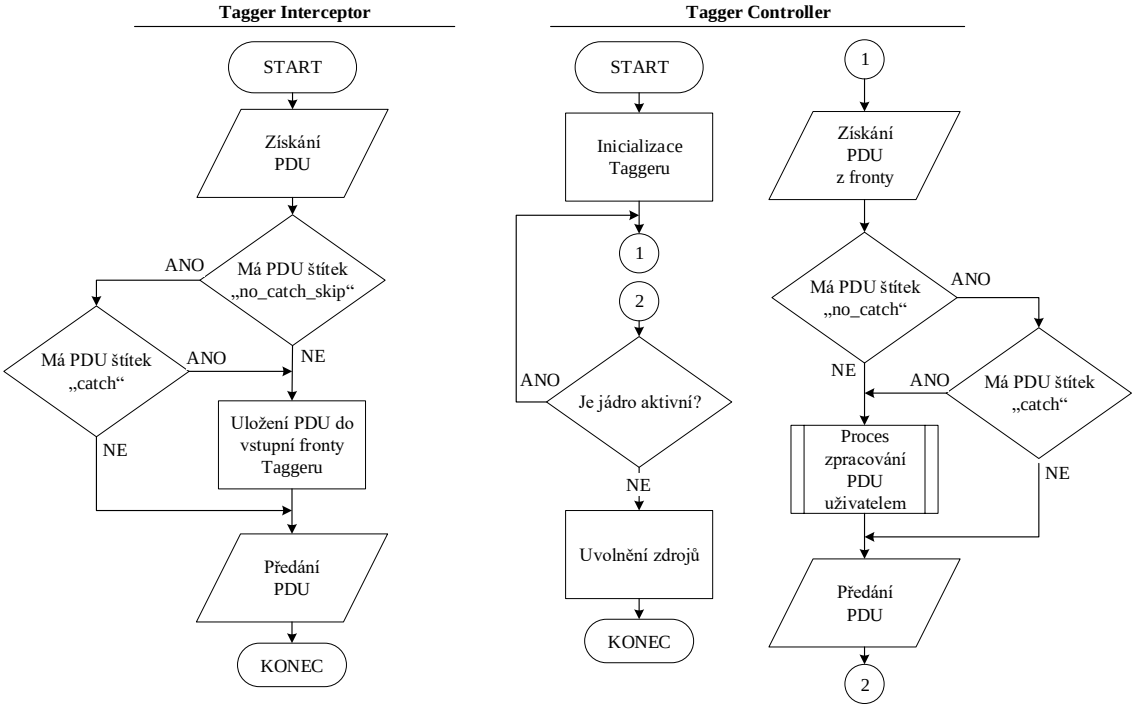
Obrázek č. 40 Pravidlo pro nahrazení části dat PDU
(Zdroj: Vlastní zpracování)

Na výše uvedeném snímku obrazovky se nachází konfigurace pravidla, které provede automatické nahrazení všech řetězců „Index of“ řetězcem „Index XYZ“, a to ve všech PDU, která budou nadřazenou modifikační skupinou zpracovávány.

3. 6. 4 Rozšíření Catcher

Rozšíření Catcher je pojmenováno na základě přeneseného významu slovíčka „chytač“, který má symbolizovat funkcionalitu spočívající v zachytávání PDU procházejících skrze aplikaci. Jde prakticky o interceptor modul zpřístupněný uživateli pro manuální editaci datových jednotek. Jeho základem je editor PDU, do nějž jsou načítány jednotlivé PDU z fronty, do které se řadí při svém příchodu do modulu. Uživatel může ručně provádět libovolné úpravy těchto protokolových datových jednotek nebo je ručně propouštět či zahazovat.

Pro usnadnění práce s datovým tokem může uživatel využít tří štítků, které umožní definovat, jaké PDU propouštět a jakým způsobem (no_catch, no_catch_skip a catch). Celý princip použití speciálních štítků je popsán pomocí následujícího vývojového diagramu:



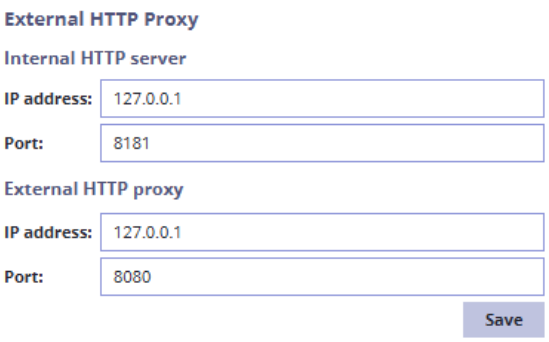
Obrázek č. 41 Vývojový diagram zpracování PDU pomocí Taggeru
(Zdroj: Vlastní zpracování)

Při použití „no_catch“ je na rozdíl od „no_catch_skip“ zachováno pořadí PDU, jelikož jsou řazeny do fronty pro editor a propuštěny až ve chvíli, kdy mělo dojít k jejich manuální editaci uživatelem.

3. 6. 5 Rozšíření External HTTP Proxy

V analýze současného stavu jsem popsal existenci několika velmi kvalitních řešení pro penetrační testování komunikace používající protokol HTTP, která jsou v praxi hojně využívána po celém světě. Při tvorbě univerzálního (na protokolu nezávislého) nástroje jsem se tak rozhodl implementovat rozšíření, jež umožní použití HTTP proxy pro práci s daty probíhajícími skrze vyvíjený nástroj.

Princip, na kterém zakládám rozšiřitelnost nástroje o externí HTTP proxy, spočívá v zapouzdření přenášených bajtů do HTTP formátu. Takto vzniklé požadavky jsou interním HTTP klientem zasílány na interní HTTP server přes externí proxy.



Obrázek č. 42 Konfigurace externí HTTP proxy
(Zdroj: Vlastní zpracování)

Účelem interního HTTP klienta je zabalit PDU přijaté z jádra aplikace do HTTP struktury a zaslat zapouzdřená data na externí proxy definovanou uživatelem. To pro rozšíření znamená, že musí být provedena serializace všech dílčích částí PDU do textové formy a výsledná data je nutné zaslat ve standardizovaném formátu směrem na externí proxy, která je po zpracování zašle internímu HTTP serveru.

```
POST /https_wss/11/4/s HTTP/1.0
T: tag_1,tag_2
M-RSV: 0,0,0
M-Mask: 79 56 6B 38
M-Opcode: TEXT
M-Fin: 1
M-Masked: 1
Content-Length: 18
Content-Type: text/plain; charset=UTF-8

{"content":"test"}
```

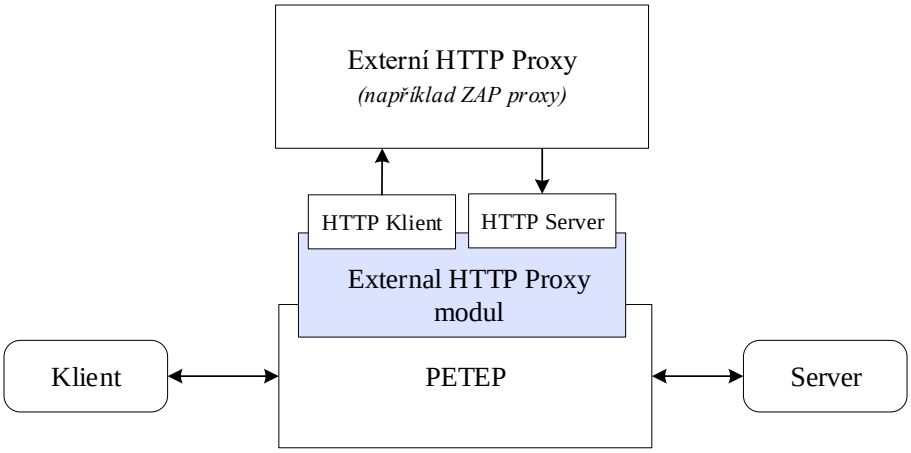
Obrázek č. 43 PDU zapouzdřené do HTTP požadavku
(Zdroj: Vlastní zpracování)

Vzniklý požadavek obsahuje hlavičku pro tagy oddělenými čárkou (T), hlavičky pro metadata s prefixem (M-), hlavičky nutné pro správný přenos dat, samotnou datovou část a cestu ve formátu:

/[proxy_code]/[id_connection]/[target_interceptor_id]/[destination]

Účelem HTTP serveru je vyextrahovat z přijatého požadavku příslušná data a na jejich základě vytvořit PDU, které následně zašle k dalšímu zpracování zpět do jádra aplikace.

Mezi klientem a serverem se tak nachází HTTP proxy, která uživateli umožňuje jak modifikovat existující protokolové datové jednotky, tak vytvářet nové, a to vše v pohodlí velmi dobře zpracovaných externích nástrojů, které jsou testeři zvyklí používat.



Obrázek č. 44 Schéma zapojení externí HTTP proxy pro zpracování dat
(Zdroj: Vlastní zpracování)

Při použití v praxi lze využít externí nástroje například pro provádění fuzz testování, hledání chyb v byznys logice či testování přetečení bufferu.

3. 6. 6 Rozšíření Logger

Logování dat poskytované rozšířením Logger je důležité pro zpětné nahlížení do dat procházejících přes proxy a lze ho využít pro analýzu a pochopení probíhající komunikace. Zpracovávané jednotky PDU jsou serializovány do člověkem čitelného textového formátu a uloženy do uživatelem specifikovaného souboru. Jednotlivé záznamy jsou v logovacím souboru odděleny oddělovači a novými řádky.

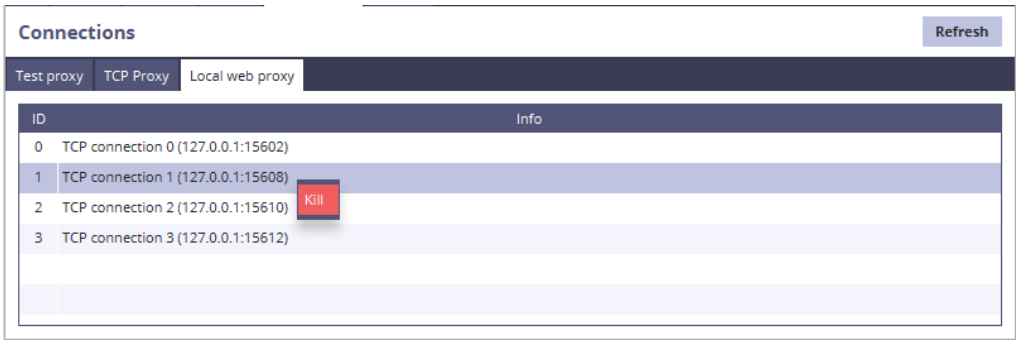
```
################################################
Time: 09/12/2019 17:42:26
Proxy: test
Destination: server
Connection: 1
Tags: test_tag_1,test_tag_2
Meta data:
   Test: test_string
Data [bytes]:
------------------------------
74 65 73 74 20 78 79 7A
------------------------------
Data [string]:
------------------------------
test xyz
------------------------------
```

Obrázek č. 45 Ukázkový záznam v logu
(Zdroj: Vlastní zpracování)

Výchozí nastavení zaručuje logování veškeré probíhající komunikace a uživatel může toto chování upravit pomocí značky „nolog“, která zamezí uložení daného balíčku, popřípadě značky „log“, která vynutí uložení daného balíčku i v případě přítomnosti značky „nolog“.

3. 6. 7 Rozšíření ConnectionView

Rozšíření ConnectionView vytváří grafický přehled o aktuálně otevřených spojeních, která jsou používána jednotlivými proxy moduly běžícími v aplikaci. Zároveň lze prostřednictvím grafického rozhraní vynutit ukončení jednotlivých spojení nebo ukončit najednou všechna spojení pro vybranou proxy.

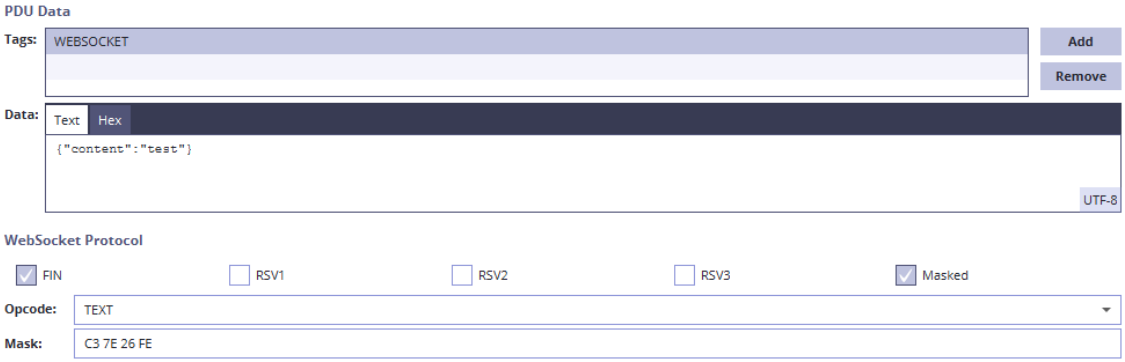


Obrázek č. 46 Zobrazení otevřených spojení
(Zdroj: Vlastní zpracování)

3. 6. 8 Rozšíření HTTP

Rozšíření HTTP zavádí do aplikace nový typ modulu proxy podporující zpracování základní HTTP komunikace, a to včetně podpory pro navázání komunikačního spojení protokolu WebSocket.

Datový obsah protokolů lze následně velmi snadno modifikovat prostřednictvím vestavěných rozšíření či externích nástrojů pro penetrační testování. Uživatel navíc získává možnost modifikovat základní parametry protokolu WebSocket, kterými jsou bit FIN, bity RSV1, RSV2, RSV3, bit MASK, 4bitový opkód a 4bajtová maska.



Obrázek č. 47 Editor PDU rozšířený o WebSocket protokol
(Zdroj: Vlastní zpracování)

Klíčovou vlastností rozšíření HTTP je jeho vybudování nad rozšířením pro TCP komunikaci, které do něj automaticky zavádí podporu pro konfiguraci obousměrného šifrování, jež bylo popsáno v kapitole o rozšíření TCP.

Pro případ přenosu příliš rozsáhlých dat, která přesahují uživatelem nastavené limity pro velikost bufferu, dojde k automatickému převodu HTTP požadavků i odpovědí na přenosové kódování zvané „Chunked Transfer Encoding“ a data jsou skrze aplikaci zasílána po jednotlivých úryvcích o maximální délce dané uživatelskou konfigurací.

3. 7 Webová prezentace aplikace

Klíčovou vlastností aplikace je její rozšiřitelnost, pro kterou je nutné zavést alespoň základní dokumentaci pro potenciální vývojáře, a zároveň je také vhodné vytvořit místo, na němž mohou uživatelé nalézt základní informace a návody a na němž mohou aplikaci získat. Pro tyto účely jsem se rozhodl vytvořit webovou prezentaci, která bude obsahovat zejména následující prvky:

- základní informace o aplikaci,
- sekci pro stažení aplikace,
- příručku pro uživatele,
- příručku pro vývojáře,
- kontaktní formulář.

Stejně jako v případě aplikace, i v případě webové prezentace jsem kvůli cílové skupině uživatelů zvolil anglický jazyk a pro grafickou ucelenost jsem zvolil totožné odstíny barev, které jsou používány v celé aplikaci. Výsledná webová prezentace je také plně responzivní a lze ji bez problému procházet i na mobilních zařízeních.

Snímky z webové prezentace jsou přiloženy v příloze číslo I a II.

3. 7. 1 Uživatelská příručka

Uživatelská příručka ve webovém rozhraní popisuje veškeré základní i pokročilejší funkce aplikace z uživatelského pohledu a do jisté míry vychází z textací v interní příručce aplikace. Hlavním cílem je seznámit uživatele se základy aplikace, vysvětlit jim použitou architekturu a terminologii, se kterou se v aplikaci operuje, a následně je naučit využívat jednotlivé části aplikace.

3. 7. 2 Vývojářská příručka

Vývojářská příručka ve webovém rozhraní je oproti příručce uživatelské podstatně rozsáhlejší a v první části vysvětluje vývojářům, jakým způsobem mohou rozšířit aplikaci o vlastní funkcionalitu. Ve druhé části se příručka zaměřuje na vývoj ukázkového rozšíření, které využije většinu existujících rozhraní a může být velmi snadno použito jako výchozí bod při psaní nového rozšíření externím vývojářem.

Součástí příručky pro vývojáře jsou úryvky i kompletní zdrojové kódy se zvýrazněnou syntaxí, které dávají vývojáři praktický přehled o popisované problematice. Dále jsou zde vypsána konkrétní doporučení, kterých by se měl programátor při tvorbě svých rozšíření držet, a varování udávající, na co by si měl dát vývojář při tvorbě svých rozšíření pozor.

3. 8 Ekonomické zhodnocení

Hlavním cílem práce bylo navržení a vývoj proxy aplikace pro účely penetračního testování, které usnadní provádění testů tlustých klientů. Z pohledu nákladů na výsledný produkt tak jde zejména o náklady na čas potřebný k návrhu a vývoji aplikace a další náklady rozepsané v následující tabulce:

Tabulka č. 1 Náklady projektu
(Zdroj: Vlastní zpracování)

Náklad	Pracnost (MD)	Cena (Kč)
Návrh a vývoj aplikace	40	600 000,-
Vývoj webové prezentace	3	45 000,-
Sepsání textací	2	30 000,-

Výsledné celkové náklady aplikace, při ilustrační částce 15 tisíc Kč/MD, dosahují výše 675 000 Kč.

3. 9 Přínosy aplikace

Přínosy aplikace lze hodnotit ze dvou úhlů pohledu, a to z pohledu prospěšnosti pro bezpečnostní specialisty, kteří budou výsledný produkt používat, a z pohledu významu pro rozšíření mého vlastního portfolia.

Bezpečnostním expertům výsledný produkt přináší nové možnosti a napravuje nedostatky existujících řešení, které byly zanalyzovány ve druhé kapitole. Díky tomuto produktu tak mohou bezpečnostní specialisté poskytovat kvalitnější služby a efektivněji využívat čas při provádění penetračních testů.

Druhý pohled na zhodnocení přínosů je, že návrh a vývoj aplikace PETEP je pro mne určitou formou investice vlastního volného času do rozvoje dovedností, a zároveň do budování osobního projektu, jenž je cennou referencí, o kterou mohu rozšířit své osobní portfolio.

ZÁVĚR

Cílem této práce bylo navrhnout a vytvořit proxy pro penetrační testování, která bude sloužit jako kvalitní podpůrný nástroj pro bezpečnostní specialisty zaměřující se na hledání bezpečnostních zranitelností v tlustých klientech a jim příslušejících serverových službách.

Celá bakalářská práce se opírá o teoretické pozadí z oboru počítačových sítí, vývoje aplikací a bezpečnosti informačních technologií. Nejdůležitější pojmy a teoretická východiska, bez nichž by nebylo možné práci zpracovat, jsem popsal v první kapitole na základě rozmanitých odborných zdrojů. Odlišnost výsledného produktu od existujících řešení je postavena na hlavních vlastnostech řešení, jež vychází z analýzy současného stavu sestávající z analýzy současné situace testování tlustých klientů, analýzy používaných operačních systémů z hlediska penetračních testerů i testovaných aplikací a analýzy konkurenčních aplikací.

Výstupem práce je funkční aplikace splňující všechny požadované vlastnosti a webová prezentace sloužící pro propagaci a distribuci současných i budoucích verzí vyvinuté aplikace. Pro samotnou aplikaci byl zvolen název PETEP vycházející z anglického Penetration Testing Proxy. Mezi hlavní přednosti aplikace patří zejména její rozšiřitelnost spojená s modularitou, která poskytuje rozhraní pro doplňování nové funkcionality, a to pod silnou vrstvou abstrakce, jež podstatně usnadní integraci nových síťových protokolů, tvorbu speciálních modifikačních pravidel nebo například vývoj modulů pro komplexní zpracování protokolových datových jednotek.

Další výhody vyvinutého řešení spočívají v pokrytí většiny používaných operačních systémů, přívětivém grafickém rozhraní a podpoře více paralelně otevřených spojení i více paralelně spuštěných proxy serverů. Celý nástroj lze navíc snadno napojit na existující nástroje vytvořené pro testování aplikací používajících protokol HTTP.

Součástí nástroje je mimo jiné osm interních rozšíření, která přidávají nejdůležitější funkce pro práci se síťovou komunikací, a to zejména automatické označování PDU na základě uživatelem stanovených pravidel, automatizované editování datového obsahu a podporu pro protokol TCP. Důležitou komponentou poskytovaného řešení je také kompletní uživatelská a vývojářská příručka v anglickém jazyce, kterou mohou využít vývojáři třetích stran pro vytváření vlastních rozšíření.

SEZNAM POUŽITÝCH ZDROJŮ

(1) SCHILDT, Herbert. *Mistrovství - Java*. 1. vydání. Brno: Computer Press, 2014. Mistrovství. ISBN 978-80-251-4145-8.

(2) *JavaFX* [online]. Leuven: Gluon HQ, 2019 [cit. 2019-10-12]. Dostupné z: https://openjfx.io/

(3) GOOGLE INC. *Gson* [online]. 2008 [cit. 2019-10-12]. Dostupné z: https://github.com/google/gson/

(4) *Gradle* [online]. San Francisco: Gradle Inc., 2019 [cit. 2019-10-12]. Dostupné z: https://gradle.org/

(5) *Extensible Markup Language (XML) 1.0* [online]. 5th ed. Cambridge: W3C, 2008, 7 February 2013 [cit. 2019-10-31]. Dostupné z: https://www.w3.org/TR/xml/

(6) *HTML Standard* [online]. WHATWG, 2019, 31 October 2019 [cit. 2019-11-01]. Dostupné z: https://html.spec.whatwg.org/

(7) *CSS Tutorial* [online]. Sandnes: Refsnes Data, c1999-2019 [cit. 2019-11-01]. Dostupné z: https://www.w3schools.com/css/

(8) SOSINSKY, Barrie A. *Mistrovství - počítačové sítě: [vše, co potřebujete vědět o správě sítí]*. Brno: Computer Press, 2010. ISBN 978-80-251-3363-7.

(9) KABELOVÁ, Alena a Libor DOSTÁLEK. *Velký průvodce protokoly TCP/IP a systémem DNS*. 5., aktualiz. vyd. Brno: Computer Press, 2008. ISBN 978-80-251-2236-5.

(10) FETTE, I a A MELNIKOV. GOOGLE, INC. *The WebSocket Protocol*. California: IETF, 2011. ISSN 2070-1721. Dostupné také z: https://tools.ietf.org/html/rfc6455

(11) WEIDMAN, Georgia. *Penetration testing: a hands-on introduction to hacking*. 1st Edition. San Francisco: No Starch Press, 2014. ISBN 978-1593275648.

(12) *The Penetration Testing Execution Standard* [online]. b.r., 30 April 2012 [cit. 2019-11-01].

(13) KIM, Peter. *The hacker playbook 2: practical guide to penetration testing*. North Charleston (SC): Secure Planet, 2015. ISBN 978-1512214567.

(14) *Infosec Resources: IT Security Training & Resources* [online]. Madison: Infosec, 2019 [cit. 2019-12-06]. Dostupné z: https://resources.infosecinstitute.com/

(15) KUMAR, Vinay. Advice from a Real Hacker: Top 10 Advanced Operating Systems For Hacking. In: *Pro Hacker* [online]. Prophet Hacker, 2016 [cit. 2020-01-23]. Dostupné z: https://www.prophethacker.com/2016/06/best-os-hacking-2106.html

(16) Operating System Market Share Worldwide. *StatCounter Global Stats* [online]. Dublin: StatCounter, c1999-2020 [cit. 2020-01-19]. Dostupné z: https://gs.statcounter.com/os-market-share

(17) *Wireshark* [online]. b.r. [cit. 2020-01-23]. Dostupné z: https://www.wireshark.org/

(18) Echo Mirage. *SourceForge* [online]. Slashdot Media, b.r. [cit. 2020-01-23]. Dostupné z: https://sourceforge.net/projects/echomirage.oldbutgold.p/

(19) Mallory. *GitHub* [online]. San Francisco: GitHub, Inc., 2020 [cit. 2020-01-23]. Dostupné z: https://github.com/intrepidusgroup/mallory

(20) *OWASP ZAP* [online]. Maryland: OWASP, 2020 [cit. 2020-01-23]. Dostupné z: https://www.zaproxy.org/

(21) Burp Suite: Cybersecurity Software from PortSwigger. *PortSwigger* [online]. Knutsford: PortSwigger Ltd., 2020 [cit. 2020-01-23]. Dostupné z: https://portswigger.net/burp

SEZNAM POUŽITÝCH OBRÁZKŮ

SEZNAM POUŽITÝCH GRAFŮ

SEZNAM POUŽITÝCH TABULEK

SEZNAM POUŽITÝCH ZKRATEK A SYMBOLŮ

TCP – Transmission Control Protocol

IP – Internet Protocol

JVM – Java Virtual Machine

OOP – Object-Oriented Programming

I/O – Input / Output

MITM – Man In The Middle

PDU – Protocol Data Unit

JAR – Java Archive

GUI – Graphical User Interface

XML – eXtensible Markup Language

API – Application Programming Interface

JSON – JavaScript Object Notation

HTML – HyperText Markup Language

HTTP(S) – Hypertext Transfer Protocol (Secure)

SSL – Secure Sockets Layer

TLS – Transport Layer Security

CSS – Cascading Style Sheets

PTES – Penetration Testing Execution Standard

ISO – International Organization for Standardization

OSI – Open Systems Interconnection

ASCII – American Standard Code for Information Interchange

SMTP – Simple Mail Transfer Protocol

POP3 – Post Office Protocol 3

IMAP – Internet Message Access Protocol

VLAN – Virtual Local Area Network

NDP – Neighbor Discovery Protocol

QoS – Quality of Service

UDP – User Datagram Protocol

IoT – Internet of Things

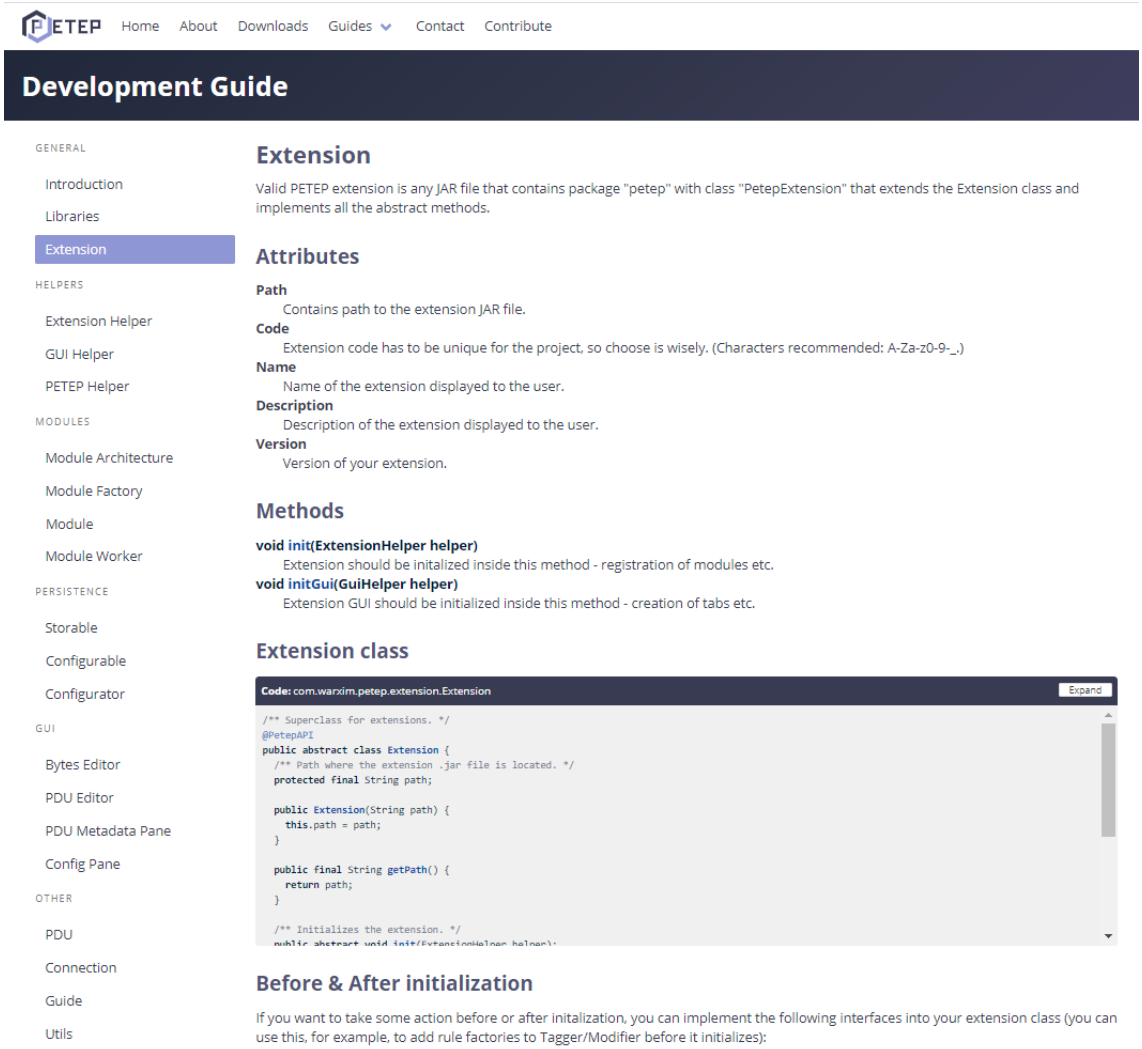
DNS – Domain Name System

FTP – File Transfer Protocol

ICMP – Internet Control Message Protocol

SEZNAM PŘÍLOH

PŘÍLOHA Č. I: Webová prezentace na desktopu (vývojářská příručka)



PŘÍLOHA Č. II: Webová prezentace v mobilním zařízení (kontakt)